Fuzzy Logic Based FPGA Routing

by

Talal Mousa Mohammed Al-Kharobi

A Thesis Presented to the

FACULTY OF THE COLLEGE OF GRADUATE STUDIES

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN, SAUDI ARABIA

In Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

In

COMPUTER ENGINEERING

June, 1998

INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

UMI

A Bell & Howell Information Company
300 North Zeeb Road, Ann Arbor MI 48106-1346 USA
313/761-4700 800/521-0600

FUZZY LOGIC BASED FPGA ROUTING

BY

Talal Mousa Mohammed Al-Kharobi

A Thesis Presented to the

FACULTY OF THE COLLEGE OF GRADUATE STUDIES

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN, SAUDI ARABIA

In Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE
In
COMPUTER ENGINEERING

June 1998

KING FAHD UNIVERSITY OF PETROLEUM AND MINERALS
DHAHRAN 31261, SAUDI ARABIA

COLLEGE OF GRADUATE STUDIES

This thesis, written by *Talal Mousa Mohammed Al-Kharobi* under the direction of his Thesis Advisor and approved by his Thesis Committee, has been presented to and accepted by the Dean of the College of Graduate Studies, in partial fulfilment of the requirements for the degree of MASTER OF SCIENCE IN COMPUTER ENGINEERING.

Thesis Committee

Dr. Habib Youssef
(Chairman)

Dr. Alaaeldin Amin
(Co-Chairman)

Dr. Sadiq Sait
(Member)

Dr. Khaled Al-Tawil
(Department Chairman)

Dr. Abdullah Al-Shehri
Dean, College of Graduate Studies

COLLEGE OF GRADUATE STUDIES

27/5/98
Date

DEDICATION

to my parents

ACKNOWLEDGEMENT

Acknowledgment is due to King Fahd University of Petroleum & Minerals and King Abdul-Aziz City of Science and Technology for support of this research.

I wish to express my appreciation to my Thesis Committee Dr. Habib Youssef (Chairman), Dr. Alaaeldin Amin (Co-Chairman), and Dr. Sadiq Sait (Member).

CONTENTS

LIST OF TABLES

LIST OF FIGURES

خلاصة الرسالة

الاسم طلال موسى محمد الخروبي

عنوان الرسالة التوصيل في مصفوفات البوابات المنطقية القابلة للبرمجة الحقلية باستخدام المنطق الغامض

التخصص هندسة الحاسب الآلي

تاريخ التخرج صفر ١٤١٩هـ

مصفوفات البوابات المنطقية القابلة للبرمجة الحقلية (FPGAs) أداة مفيدة جداً لتصميم الدوائر المتكاملة ذات التطبيقات المحددة وإنتاج نماذج التصاميم في وقتٍ قصير. استخدام مصفوفات البوابات المنطقية لتصميم معين يمر بعدة خطوات. نظراً لأن مصفوفات البوابات المنطقية معقدة نسبياً، كان من المحتم أتمتة هذه الخطوات. ويعتبر التوصيل الداخلي آخر خطوة من هذه الخطوات والتي يتم بعدها برمجة المصفوفة. في هذه الأطروحة، تم تصميم موصل داخلي لفصيلة جديدة من المصفوفات المنطقية ذات بنية جزيرّية (أو متماثلة). تتضمن الأطروحة تفاصيل هذا التصميم الجديد وخوارزمية التوصيل ضمن هذا التصميم. الخوارزمية تقسم عملية التوصيل إلى التوصيل المجمل والتوصيل المفصل. التوصيل المجمل يتبع خوارزمية "Lee" للتوصيل ضمن متاهة. تستند خوارزمية التوصيل المجمل على المنطق الغامض لتقييم الحلول المختلفة اعتماداً على عدة متطلبات. التوصيل المفصل يعتمد على خوارزمية توسيع الرسم ذو الخلايا العريضة (CGE) ويتم تقييم الحلول المختلفة باستخدام المنطق الغامض أيضاً. تم برمجة واختبار الموصل الداخلي بجزأيه المجمل والمفصل.

درجة الماجستير في العلوم

جامعة الملك فهد للبترول والمعادن

الظهران - المملكة العربية السعودية

صفر ١٤١٩هـ

THESIS ABSTRACT

NAME *Talal Mousa Mohammed Al-Kharobi*

TITLE *Fuzzy Logic Based FPGA Router*

MAJOR FIELD *Computer Engineering*

DATE OF DEGREE *June 1998*

Field programmable gate arrays (FPGAs) are very useful devices for rapid ASICs design and fast prototyping. Mapping a specific design to a given FPGA passes through a number of stages. Being a fairly complex device, the design process must be automated. A separate computer program is written to perform each stage. Routing is the last phase in the design cycle. In this thesis, a router for a new family of island-based FPGAs is developed. The architecture of the proposed FPGA is described in detail, and the algorithm to route nets within such structure is outlined. The algorithm divides the routing problem into two steps: global routing and detailed routing. The global router follows a variation of the Lee maze routing algorithm and uses fuzzy logic to evaluate routing alternatives based on several objectives. The detailed router uses a coarse graph expansion (CGE) algorithm with a fuzzified cost function. Both the global and detailed router algorithms were successfully implemented and tested.

MASTER OF SCIENCE DEGREE
KING FAHD UNIVERSITY OF PETROLEUM AND MINERALS
Dhahran, Saudi Arabia
June 1998

CHAPTER 1

INTRODUCTION

Rapid improvement of VLSI design methodology and tools has reduced the cost of designing digital circuits and allowed integrating millions of transistors on a single chip. This has resulted in an increased demand for application specific integrated circuits (ASICs). On the other hand, building a VLSI chip is cost effective only if it is produced in large volumes. This is due to the tremendous time and effort needed for its design. To produce ASICs, one of the following methods is used:

1. Full custom design.
2. standard cells design,
3. gate arrays design, or
4. field programmable devices.

In the full custom design method, all circuit modules are custom-designed and their layout manually generated. Thus, custom designed ICs exhibit high performance at the expense of high cost and long turn-around-time (TAT).

The standard cells design approach uses a set of pre-designed and pre-characterized

cells to build the required system. Mask programmable gate array (MPGA) is an array of transistors pre-fabricated up to the last 2 to 3 masks. The manufacturer can easily configure this array to a specific design. Both types of designs (the standard cells and MPGAs) require about 6 weeks to produce a chip and they cost a lot less than full-custom design [13].

Field programmable devices (FPDs) are such devices that can be configured by the end user (in field) without the need for any integrated circuit fabrication facility. The time to produce a design is few minutes to few hours and at a very low cost [16].

For ASICs, the production values are typically smaller than that of standard integrated circuits. In addition, for prototypes, relatively small quantities are required to test the design before large volume production. For ASIC or prototyping, designs need to get to the market in reasonably short time and at reasonably low cost. Therefore, the production of such chips is done using programmable devices.

Programmable devices can be either mask or field programmable. The mask programmable devices are programmed by the manufacturer and the connections are hardwired, while, the field programmable devices are programmed by the user and involve field programmable switches. Mask programmable devices have better performance, while, field programmable ones have faster TAT and lower cost. In addition, field programmable devices can provide the user with more privacy [16].

The scope of this work is to develop a router for a new family of island-based field programmable gate arrays (a type of field programmable devices). Next, some highlights on routing VLSI circuits are given. Then, field programmable gate arrays (FPGAs) are introduced. A discussion on FPGA routing will follow. Finally, thesis organization is given.

1.1 Routing

Routing is the final step in the design of ASICs. The objective of routing is to find, for each net in a given circuit, a path that connects all net pins. In some design technologies, the router is responsible of estimating channels locations and widths. In others, e.g. FPGAs, the router should find paths to connect all the nets in the design within the available routing resources. Routing consumes about 30% of design time. Interconnection resources usually occupy almost 40% of the chip area [18].

Being a fairly complex process, routing is usually automated. A computer program (called router) is written to interconnect all nets of a given circuit. The inputs to the router are (1) the set of nets, (2) circuit layout and (3) routing resources architecture. The router should provide suitable paths for connecting all nets within the available routing resources. For multi-pin nets, the router should find a tree that connects all net

pins. Such tree could be a spanning or a Steiner tree [18].

To simplify the problem, routing is usually performed in two steps: global routing, and detailed routing. Some details on both global and detailed routers are given next. First, the Steiner tree problem is introduced.

1.1.1 The Steiner Tree Problem

One of the important parameters of a net, that the router should minimize, is its length. The length of a two-pin net is simply the length of the path that connects the two pins. For multi-pin nets, a tree structure that connects all net terminals together is required. The tree with the shortest total length for the net is known as the minimum Steiner tree [18]. Nodes of the Steiner tree need not necessarily be net pins (Figure 1). However, a tree that connects all net pins and all the tree nodes are the net pins only is called a spanning tree (Figure 1).

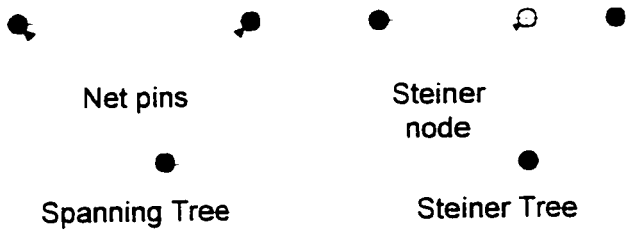


Figure 1: Spanning tree and Steiner tree.

Finding a Steiner tree that connects a set of pins is a task commonly performed by the

router. Begin an NP-hard problem, the generation of Steiner tree for the net to be routed has a profound effect in increasing the router complexity [18].

1.1.2 Global Routing

Global routing is performed first to elaborate a routing plan so that each net is assigned a specific routing region while attempting to optimize a given objective function. There are several approaches to perform global routing. These approaches can be classified into the following four categories:

1. Sequential approaches.
2. mathematical programming approaches.
3. stochastic iterative approaches. and
4. hierarchical approaches.

The sequential approach routes individual nets one at a time in a specific order. The router is order dependent if the routing region is updated after routing each net: otherwise, it is order independent [18].

In the mathematical programming approach, global routing is formulated as a 0-1-integer optimization program, where a 0-1-integer variable is assigned to each net and each possible routing tree of that net [18].

The stochastic iterative approach starts with some initial solution and iteratively

updates the current solution by ripping up and rerouting selected nets until an acceptable routing plane of the nets is found. An example of such approach is simulated annealing [18].

Hierarchical approach can be implemented either in a bottom up or in a top down manner. In bottom up schemes, the grid cells are clustered into bigger cells until the entire chip is clustered into one super cell. At each level of the hierarchy, global routing is performed between the individual cells considered for grouping. In the top down scheme, the hierarchy proceeds from super cells to smaller ones until each cell is an individual grid cell or a small group of individual grid cells. The hierarchical decomposition is usually guided by the structure of the design floorplan [18].

1.1.3 Detailed Routing

The detailed router uses the assignment of nets to routing regions (the output of the global router) to assign for each net particular tracks within these regions. In contrast to the global router, the detailed router depends heavily on the circuit architecture and routing region structure.

1.2 Field Programmable Gate Arrays

A field programmable gate array (FPGA) consists of a number of uncommitted modules that can be configured to implement a special function. FPGA configuration is done through controlling a set of programmable switches. FPGAs have been successfully used in the military, scientific and general applications [1]. They can be used for re-configurable computer interfacing. In addition, they can be used to replace some component(s) in a ready-built digital system.

According to the switch programming technology, FPGAs can be classified into:

1. SRAM[1] based FPGAs,
2. anti-fuse based FPGAs, or
3. EPROM[2] or EEPROM[3] based FPGAs [23].

In SRAM-based FPGAs, programming is done using SRAM cells that control either pass-transistors, transmission gates, or multiplexers (Figure 2) [16]. Since the data stored in the SRAM cells is volatile, the FPGA configuration data should be downloaded into the SRAM cells each time the FPGA is powered ***ON***. While SRAM-based FPGAs can be easily reprogrammed, configuration data security is not guaranteed [13].

---

[1] Static Random Access Memory
[2] Erasable Programmable Read Only Memory
[3] Electrically Erasable Programmable Read Only Memory

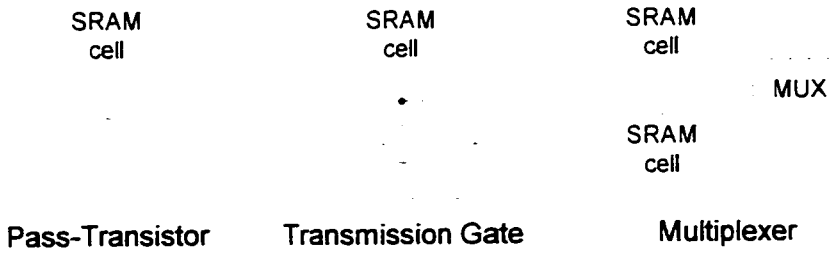


Figure 2: Static RAM programming technology.

Anti-fuse-based FPGA uses a large number of anti fuses as programmable switches. An anti-fuse is an electrical switch that is normally open. It can be closed (short circuit), when programmed, through the application of high voltage. Anti-fuse-based FPGAs are not re-configurable since anti-fuses are one-time programmable. Unlike SRAM-based, anti-fuse-based FPGAs guarantee the security of the design [13] [16].

EPROM-based and EEPROM-based FPGAs consist of a number of transistors that can be programmed ON or OFF. The main difference between EPROM-based (or EEPROM-based) and SRAM-based FPGAs is that the data stored in the EPROM-based (or EEPROM-based) FPGA is nonvolatile. Therefore, there is no need to download the configuration data each time the device is powered on. When there is a need for reconfiguration, the cells can be erased either electrically as in EEPROMs or through using ultraviolet light as in EPROMs. Accordingly, to reconfigure the EPROM-based FPGA, it should be removed from the circuit, while EEPROM-based FPGAs can be reconfigured in-circuit. It should be noted that, EEPROM transistors have about twice the area as the EPROM transistors [16].

FPGAs can, alternatively, be classified according to their architecture; the arrangement of logic modules routing channels and switches. The most common FPGA architectures are the row-based and the island-based FPGAs. Both architectures are described next.

1.2.1 Row-Based FPGA Architecture

A row-based FPGA consists of rows of logic modules separated by horizontal routing channels (Figure 3). The horizontal routing channels consist of wire segments of varying lengths running between rows of logic modules. The channel between two rows can be used to connect logic modules on these two opposite rows. When the modules to be connected are not on opposite rows, feed-through cells together with horizontal and vertical channels are used.

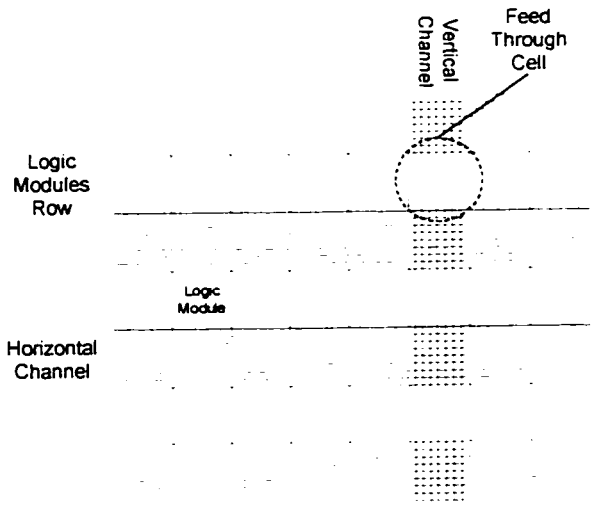


Figure 3: Row-based FPGA structure.

1.2.2 Island-Based FPGA Architecture

An island-based FPGA (sometimes called symmetrical FPGA) has a two-dimensional array of programmable logic modules separated by vertical and horizontal routing channels (Figure 4). Switch boxes which consists of a set of programmable switches are located at the intersection between vertical and horizontal channels. Logic module pins can be connected to the routing channels using another set of programmable switches, which are referred to as connection box. Thus, the array has vertical and horizontal connection boxes to connect logic modules to the vertical and horizontal channels respectively. The logic modules can be interconnected via the channels, the connection boxes and the switch boxes.

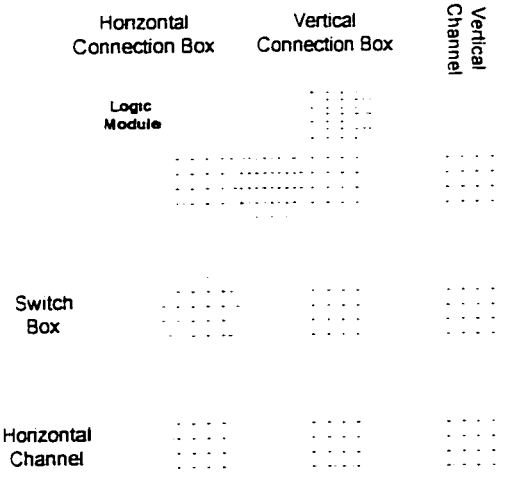


Figure 4: Island-based FPGA structure.

1.3 FPGA Routing

Mapping a given design onto specific FPGA should pass through several phases. Routing is the final phase in such mapping, after which the FPGA can be configured. Routing for FPGAs departs from the traditional routing methods due to the rigid and heterogeneous natures of the FPGA routing resources. Therefore, routing algorithms appropriate for a given FPGA depend on its routing structure. Next, routing terminology and strategy are given. Then, FPGA routing structure is discussed.

1.3.1 Terminology

A brief description of the terms used in this work, to explain the routing structure and algorithms, is given below [16].

- **Pin**: a logic module input or output.
- **Connection**: a pair of logic module pins that are to be electrically connected.
- **Net**: a set of logic module pins that are to be electrically connected.
- **Wire segment**: a straight section of wire that is used to form a part of a connection.
- **Track**: a straight section of wire that spans the entire width or length of a routing channel. A track can be occupied by a single wire segment or composed of a number of wire segments of various lengths.
- **Routing switch**: a programmable switch that is used to electrically connect two wire segments.

- **Routing channel**: the rectangular area that lies between two rows or two columns of logic modules. Each routing channel contains a number of parallel tracks.

1.3.2 Routing Strategy

Because of the complexity involved, the solution of large routing problems, such as those encountered in FPGAs, usually uses a divide and conquer strategy. Generally, the solution is divided into three steps:

1. Partition the routing resources into routing areas that are appropriate for both the device to be routed and the routing algorithm employed.
2. Perform global routing to assign each net to a subset of the routing areas. The global router does not choose specific wire segments and routing switches for each connection, but rather it creates a new set of restricted routing problems.
3. Perform detailed routing to select specific wire segments and routing switches for each connection within the restrictions set by the global router.

This strategy has been adopted for routing in the two dominant types of FPGAs (row-based and island-based). The global router first selects routing areas for each connection. Then, within the constraints imposed by the global router, the detailed router implements each connection by choosing specific wire segments and routing switches. The global routing issues are similar in both row-based and island-based FPGAs, but the detailed routing problem requires different algorithms.

Global routing might be viewed as a loose routing connecting the different logic modules pins. The FPGA global router task is similar to that used in other design styles like gate arrays and standard cells. The global router assign nets to paths, determining how will these nets navigate around the cells. These "loose" paths determine which routing areas a net will traverse. The routing areas are channels' tracks and switches. The objectives of the assignment might be to minimize the connection lengths, avoid congestion, or minimize the number of routing switches to decrease delay.

The main difference between detailed routing of traditional ICs and FPGAs is that the wire segments lengths and distribution along FPGA tracks is pre-determined. Detailed routing is heavily influenced by the architecture of the FPGA. For row-based FPGAs, routing techniques are essentially those used in traditional ASIC gate array or standard-cell design methodologies [2] with special attention to segmented tracks. In island-based FPGAs, the choice of a particular track in one channel limits the choices in the neighboring channels.

Once placed and routed, a design may not function properly due to timing problems. Rerouting with constraints may be necessary to reduce delays in order to meet the required set of given timing constraints. In delay-driven routing, the goal is to produce signal routes with reduced delays along critical paths to improve the overall circuit performance. Paths delays in an FPGA circuit has the following components:

1. Logic delay
   (a) I/O pads delay.
   (b) Logic modules delay.
2. Interconnect/routing delay
   (a) Wire segments delay.
   (b) Programmable routing switches delay.

Since delay due to routing in FPGAs accounts for 40% to 60% of the path delay [6]. delay estimation and minimization plays an important role in the FPGA design flow.

1.3.3 Routing Structure

FPGA routing depends on its routing structure. Routing structure of an FPGA is the manner in which the programmable switches and wiring segments are positioned to allow for programmable interconnection of the logic modules. Among the important design parameters which affect routability of FPGAs are: the number of tracks per routing channel, the number of segments per track, and the way tracks of various channels are connected. The routability degree of a given routing structure is referred as the structure flexibility. The structure flexibility depends on the total number of routing switches and track segments in the design. The routing structure of an FPGA includes all routing switches and wire segments, and their distribution over the surface of the chip.

Designing a good routing structure involves a tradeoff among flexibility, logic density, and speed performance. Increasing the number of switches results in higher flexibility, and hence easier routability. However, such increase in the flexibility requires more chip area and hence lower area utilization efficiency. Moreover, since each routing switch introduces an RC-delay, high flexibility results in reduced speed performance because the paths will use more switches. Low flexibility, on the other hand, allows higher logic density and lower path delay. If, however, the flexibility is too low, it may not be possible to fully route all nets using the available routing resources. Next, descriptions of routing methodology, structure, and flexibility for both row-based and island-based FPGAs are given.

1.3.3.1 Row-Based FPGA Routing

In routing row-based FPGAs, each channel can be considered as a separate detailed routing problem. A channel contains a number of connections each involving logic module pins and vertical feed-throughs. The task of the detail router is to allocate wire segments for each connection to allow all connections to be completed. One of the widely used algorithms is the *constrained left edge* algorithm [16].

The tracks in the routing channels may be fully-segmented, k-segments, or non-segmented (Figure 5). There are two types of switches in the routing channels: connection switches and segmentation switches (Figure 5). The connection switch is

used to connect a logic module pin (or a feed through cell pin) to one of the available tracks. However, the segmentation switch is used to connect/disconnect adjacent wire segments of a track. As the number of segments increases, the channels can be used to route a larger number of nets (more flexible). However, the number of switches added to segment the tracks increases the cost, area, and delay of the design.

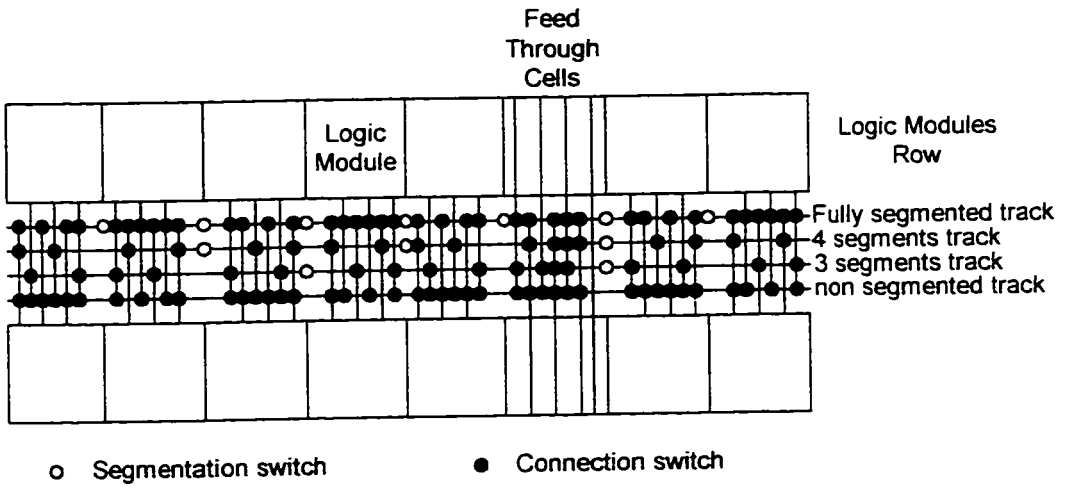


Figure 5: Routing channel structure in a row-based FPGA.

Since the router developed in this work is designed for an island-based FPGA, no farther discussion on the techniques used in row-based FPGAs routing. However, the reader is refereed to [8] [9] [15] [16] [19] [20] for details.

1.3.3.2 Island-Based FPGA Routing

Routing in island-based FPGAs is performed in two steps: global routing and detailed routing. A number of papers [11] [14] [17] [22] reported the use of some global

routing approach as used in standard cells. The global router proposed in [14] divides nets into two terminal nets and routes one pair at a time. The reported algorithm seeks to balance routing channels density. Alternatively, the router developed in [17] is a maze router and its objective is to route nets while targeting a minimized switch block congestion. In contrast, the global router derived in [22] attempts to minimize both congestion and delay. The algorithm in [11] applies A* algorithm to maze router. The global router developed in this work is maze router as well.

Detailed routers depend heavily on the design architecture i.e. the distribution and structure of logic modules, switches and routing channels. Therefore, detailed routers developed for standard cells and row-based gate arrays are not suitable. A number of techniques have been used to perform detailed routing in island-based FPGAs. The algorithm proposed in [21] is based on single net routing algorithm using a graph-theoretic approach. Number of papers uses a technique called coarse graph expansion (CGE) [2] [14] [16] [23]. This technique starts with a coarse graph provided by the global router, expands it to all possible alternatives, and chooses one of them according to some cost function. This technique is adopted in this work. Another technique uses integer liner programming is proposed in [17]. Some papers proposed algorithms to optimize existing solution [12].

The routability of island-based FPGAs depends on the flexibility of its channels, logic modules, connection boxes and switch boxes (Figure 4, page 10). Channel flexibility

depends on the number of tracks per channel and the distribution of wire segment lengths over these tracks. Tracks in the routing channels can be fully-segmented, non-segmented (long lines) or in general k-segmented. Track segments terminate at the boundaries of switch boxes, which allow connectivity to segments of other tracks (Figure 6).

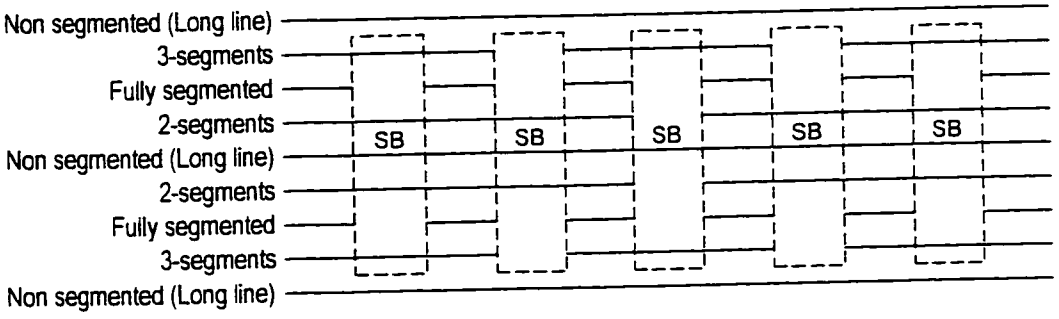


Figure 6: Horizontal channel structure in island-based FPGA.

The logic module flexibility, denoted as T, is taken to be the number of logic module sides on which each logic pin appears [3]. The maximum value of T is four (Figure 7).

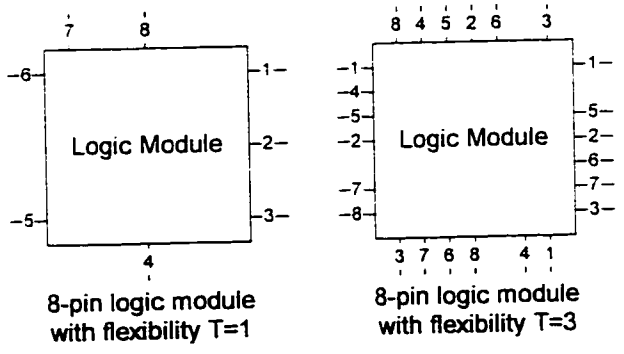


Figure 7: Logic module flexibility (T).

The connection box flexibility (FC) is the number of tracks in the adjacent channel that the logic module pins can be connected to [7]. The maximum value of FC is equal to W, the number of tracks in the channel (Figure 8).

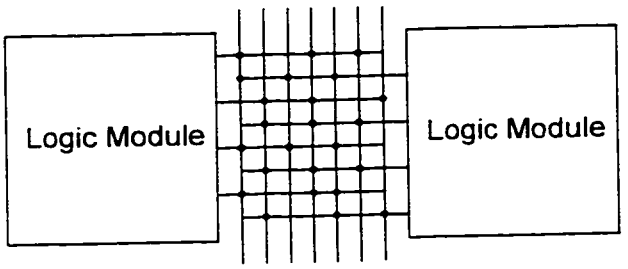


Figure 8: Vertical connection box with flexibility FC=3.

The switch box (Figure 4, page 10) provides six connection possibilities (Figure 9). The switch box flexibility (FS) is defined as the number of tracks an incoming wire can be connected to on the other three sides (Figure 10) [3] [7]. The maximum switch box flexibility is three times the channel width W.

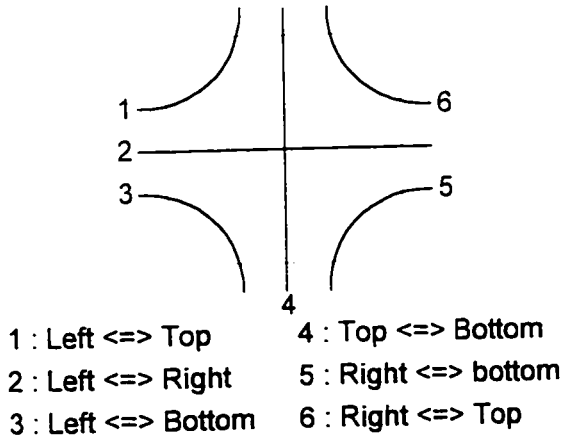


Figure 9: The six-connection possibilities configuration of a switch box.

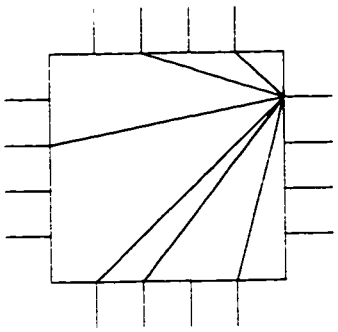


Figure 10: Switch box with flexibility FS=6.

Two types of switch boxes can be used: switch block or switch matrix (Figure 11). In a switch block, any two tracks on two different sides might have a switch between them. This switch can be programmed ON/OFF to connect/disconnect these two tracks (Figure 11-a) [10]. In the switch matrix, however, any two crossing lines can be connected by a switch. Separating switches may be used to divide the tracks within a switch matrix. The separating switches are placed on non-crossing points (Figure 11-b) [10].

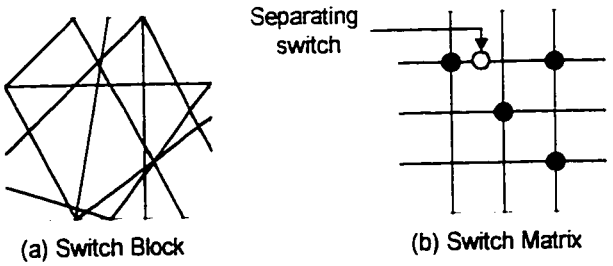


Figure 11: Two types of switch boxes.

1.4 Thesis Organization

In the following chapter, the adopted FPGA architecture is described. Next, the global routing algorithm is discussed in details and illustrated with examples (CHAPTER 3). At the end of the chapter, the global router complexity (space and time) is addressed. In CHAPTER 4, the detailed routing algorithm is discussed and analyzed. CHAPTER 5 illustrates some experimental results, observations and discussion. Finally, we conclude in CHAPTER 6.

CHAPTER 2

ADOPTED FPGA ARCHITECTURE

The objective of this work is to design a router for a new family of island-based FPGAs. The FPGA architecture adopted[1] in this work will be described in the following section. Then, the information required by the router and the format in which it should be provided are stated. The solution approach taken consists of two steps: a global routing step, and a detailed routing step. Strategies used for solving the global and detailed routing problems are presented in the following two chapters.

2.1 FPGA Architecture

The adopted FPGA consists of a 2-D array of logic modules separated by horizontal and vertical channels. Connection boxes are used to connect logic module pins to the

[1] The architecture is proposed in KACST project AR-14-67

routing channels while switch boxes are used to connect various wire segments together. The I/O pads provide connectivity between the FPGA and the outside world. There are peripheral connection and switch boxes to connect pins of the I/O pads to internal modules. The adopted FPGA architecture is shown in Figure 12. As can be seen in the figure, it consists of four main types of modules:

1. Universal Programmable Modules (UPMs)
2. Connection boxes (CBs), which can be categorized into:
   (a) horizontal CBs (HCB),
   (b) vertical CBs (VCB),
   (c) peripheral horizontal CBs (PHCB), and
   (d) peripheral vertical CBs (PVCB).
3. Switch boxes (SBs), which can be categorized into:
   (a) internal SBs (ISB),
   (b) peripheral vertical SBs (PVSB),
   (c) peripheral horizontal SBs (PHSB), and
   (d) corner SBs (CRSB).
4. Input/Output pads (I/O).

The routing channels are classified into:

1. horizontal routing channels (HRC),
2. vertical routing channels (VRC),
3. peripheral horizontal routing channels (PHRC), and
4. peripheral vertical routing channels (PVRC).

Next, details of various modules and channels are given.

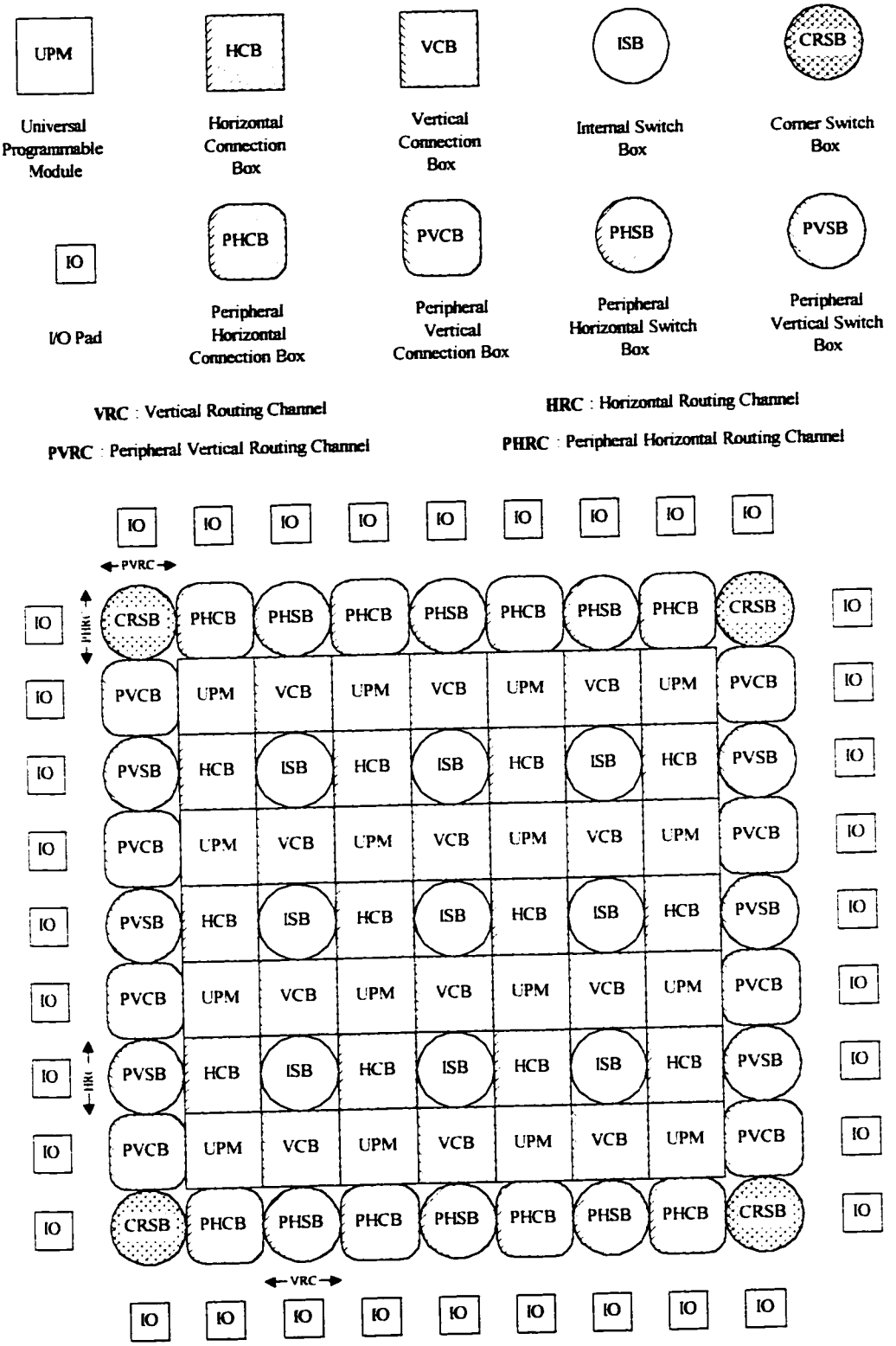
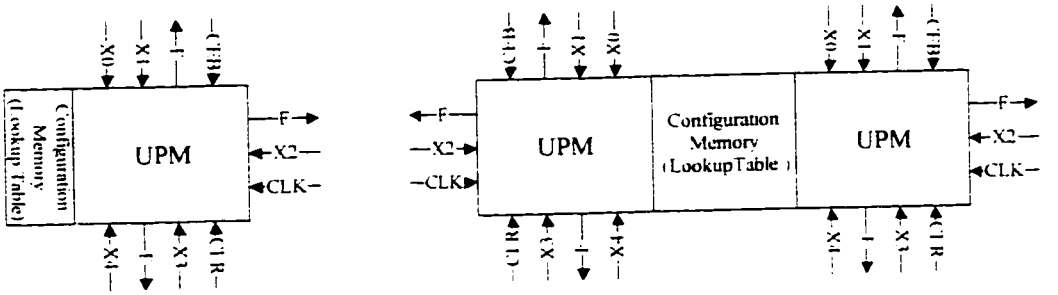


Figure 12: Adopted island-based FPGA architecture.

2.1.1 Universal Programmable Module

The universal programmable module (UPM) is a lookup table that can implement any function of five variables. Each UPM has five general inputs (X0 to X4), three flip-flop inputs (CLK, CEB and CLR), and one output F (Figure 13-a). The input pins X0 to X4 are logically equivalent. The F output is made available on three sides to improve routability (the three pins are electrically equivalent). Two UPMs can be arranged back-to-back so that they can share some of the configuration memory resources (Figure 13-b). This arrangement is adopted in this work.



(a) Single UPM (b) Two back-to-back UPMs

Figure 13: Universal programmable modules (UPMs).

2.1.2 I/O Pads

An I/O pad can be configured to function as an input pad, an output pad or a bi-directional Input-Output pad. The I/O pad consists of an input buffer, an output buffer, and control logic (Figure 14).

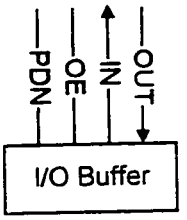


Figure 14: I/O pad in the lower FPGA side.

2.1.3 Routing Channels

Routing channel consists of a set of wire segments tracks that can be used to connect various FPGA modules' pins. The adopted FPGA has internal and peripheral routing channels. Th internal routing channels can be used to form connections between the internal modules. However, the peripheral routing channels can connect internal modules pins with peripheral modules pins. Description of each type follows.

2.1.3.1 Internal Routing Channels

Internal routing channels are located between the rows and columns of the UPMs. There are two types of routing channels: vertical and horizontal channels. The tracks in the vertical and horizontal channels are assumed symmetric with respect to the UPMs on each side (left and right sides for vertical channels and top and bottom sides for horizontal channels). Each channel has few long lines, some fully-segmented and some doubly-segmented tracks. The long lines span the full width (or length) of the channel and contain no switches. However, each fully-segmented track consists of

wire segments interrupted by all switch boxes in the channel. On the other hand, a doubly-segmented track consists of wire segments interrupted by every other switch box on the channel (Figure 15). There are two types of doubly segmented tracks even and odd. The first switch box interrupting the odd-doubly-segmented tracks is the first switch box in the row/column, while the first switch box interrupting the even-doubly-segmented track is the second switch box in the row/column (Figure 15).

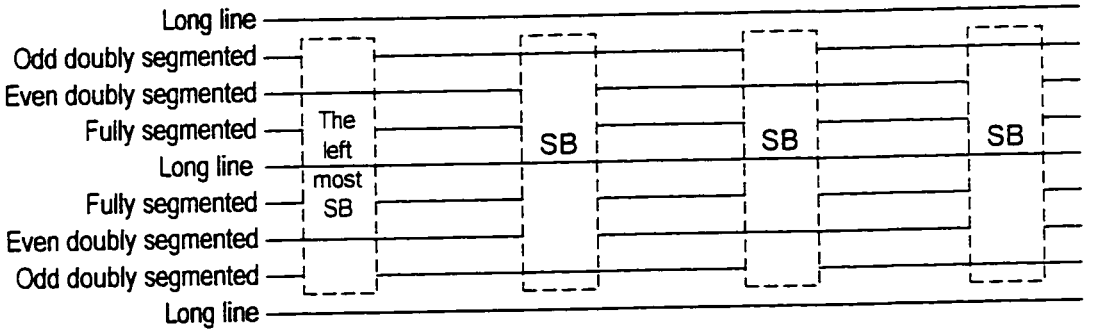


Figure 15: Internal horizontal channel.

2.1.3.2 Peripheral Routing Channels

The peripheral channels are located between the I/O pads and the peripheral UPMs. Peripheral routing channels were chosen to consist of a number of tracks that are either fully or doubly segmented (both even and odd doubly segmented tracks) as shown in Figure 16. The number of tracks in peripheral channels depends on the number of I/O pads. With equal number of pads per sides, the peripheral horizontal and vertical channels have the same number of tracks. Two long lines in the peripheral channel are dedicated for Vcc and Vss (Figure 16).

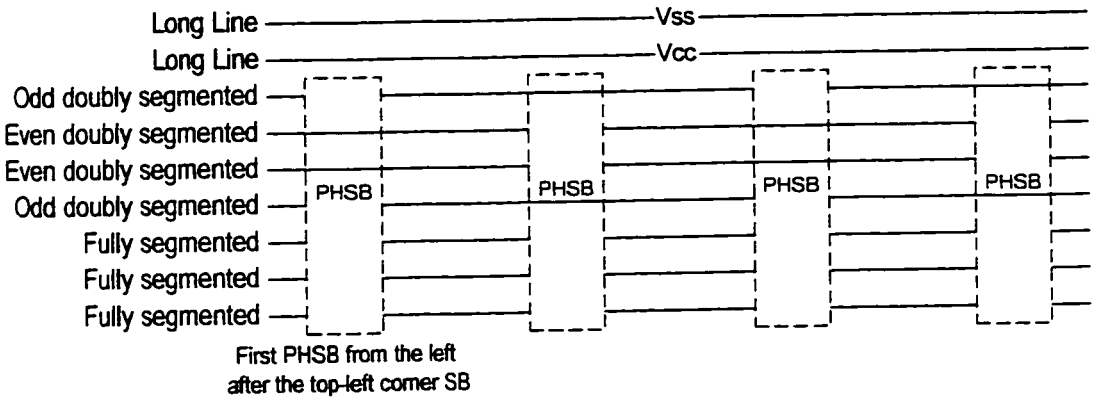


Figure 16: Peripheral horizontal channel.

2.1.4 Connection Boxes

The adopted FPGA has two internal CBs (horizontal and vertical) and two peripheral CBs (horizontal and vertical). Next, various types are described.

2.1.4.1 Internal Connection Boxes

There are two types of internal CB boxes: horizontal and vertical. The horizontal connection box connects the top and bottom pins of two opposite UPMs in the same column to the tracks of the horizontal channel between them. Each pin can be connected to a number of tracks on the channel via a set of programmable switches. Figure 17 shows an internal horizontal connection box connecting two single UPMs, while Figure 18 shows two back-to-back connection boxes connecting a pair of two back-to-back UPMs.

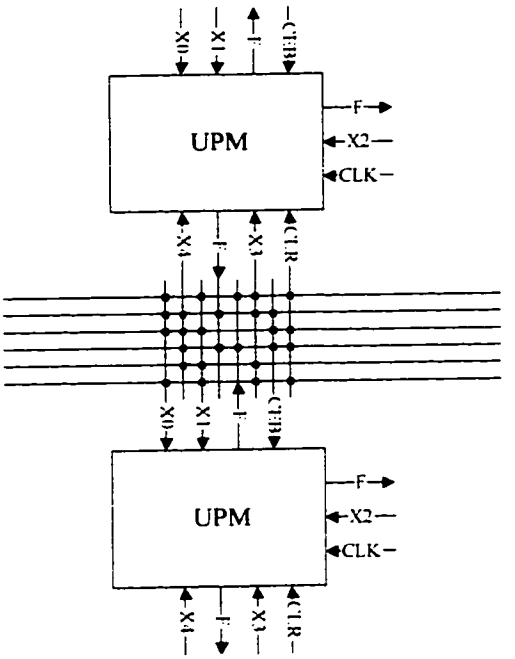


Figure 17: Internal horizontal connection box.

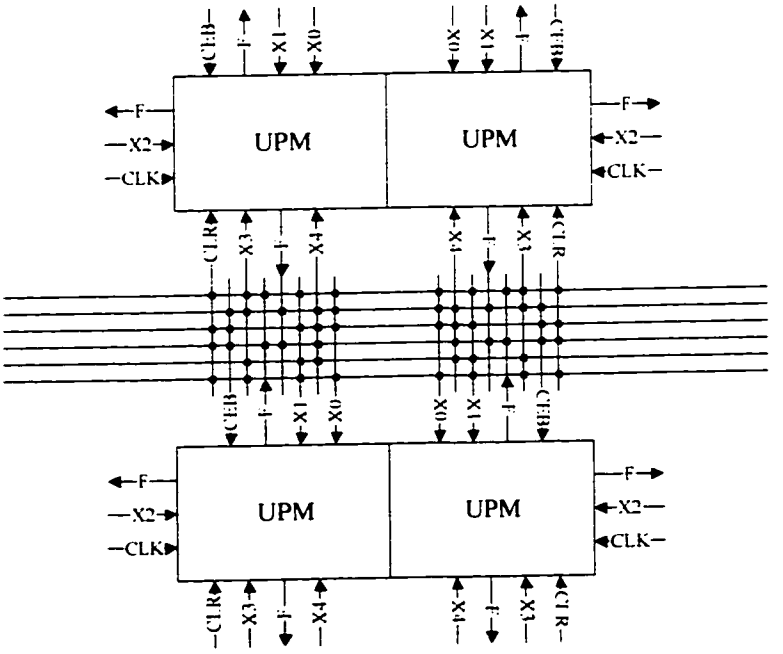


Figure 18: Two back-to-back internal horizontal connection boxes.

Vertical CBs connect the left and right pins of two opposite UPMs in the same row to

the tracks of the vertical channel between them. Each pin can be connected to several channel tracks via a set of programmable switches. A special track is dedicated to the clock network in each vertical channel. Accordingly, the UPM clock input (CLK) can only connect to this particular track (Figure 19).

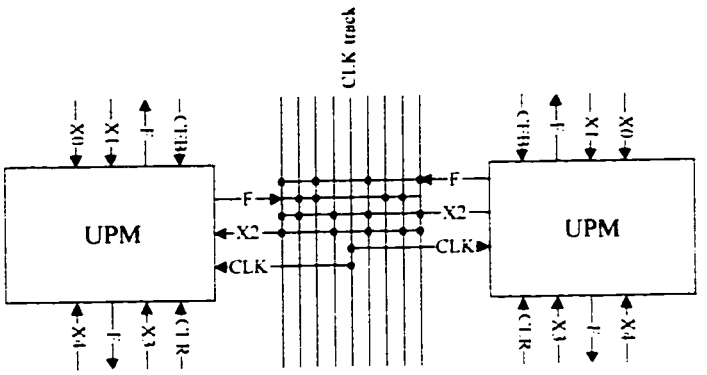


Figure 19: Internal vertical connection box.

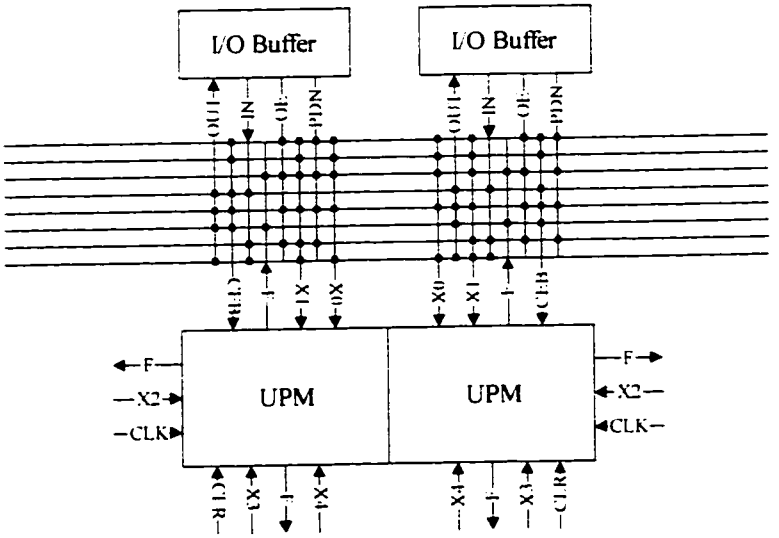


Figure 20: Two back-to-back peripheral horizontal connection boxes.

2.1.4.2 Peripheral Connection Boxes

Peripheral CB connects a peripheral UPM pins and the pins of the I/O pad facing that

UPM to the tracks of the peripheral channel between them. Figure 20 shows two back-to-back peripheral horizontal CBs on the top peripheral channel.

2.1.5 Switch Boxes

The adopted FPGA switch boxes (SBs) can be classified into three main types: internal SBs, peripheral SBs and corner SBs. Description of each type is given next.

2.1.5.1 Internal Switch Boxes

The internal SB has the capability of connecting wire segments of various internal channels. It can connect a segment coming from one side to a number of segments on the three other sides. Figure 21 shows a possible SB structure where a dotted line indicates a possible connection through a programmable switch that can be ON or OFF.

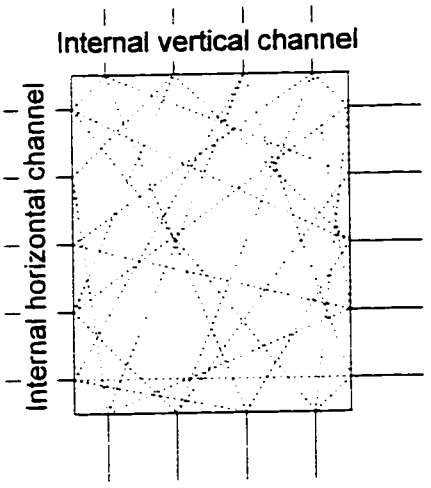


Figure 21: Internal switch box.

2.1.5.2 Peripheral Switch Boxes

Peripheral SBs connect segments of internal routing channels to tracks of peripheral routing channels. In addition, they allow connecting peripheral channel wire segments together. Figure 22 shows a horizontal peripheral SB on the top row. Peripheral SBs may be used to connect I/O pads pins to peripheral channels (Figure 23).

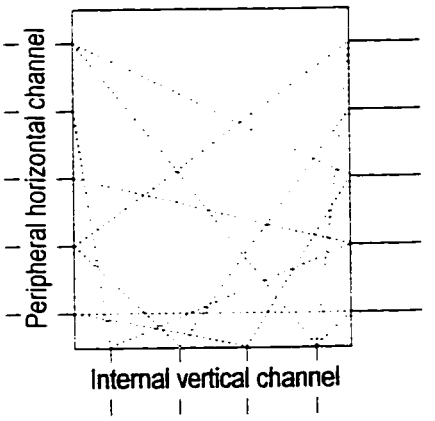


Figure 22: Peripheral horizontal SB in the top row.

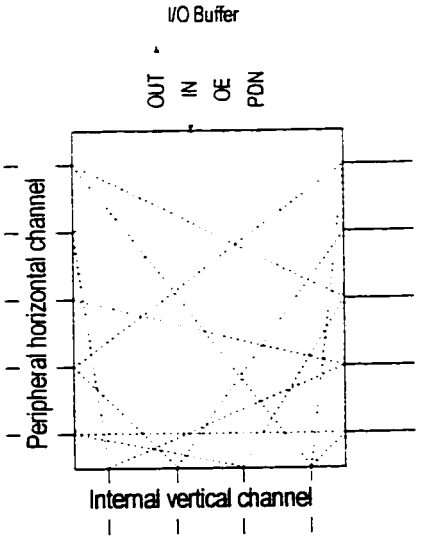


Figure 23: Peripheral horizontal SB connecting I/O pad to the top peripheral channel

2.1.5.3 Corner Switch Boxes

Corner SBs are used to connect tracks of peripheral horizontal channels to tracks of peripheral vertical channels. Figure 24 shows a possible corner SB on the top left corner. The corner SB may be used to connect two I/O pads to the tracks of peripheral channels (Figure 25).

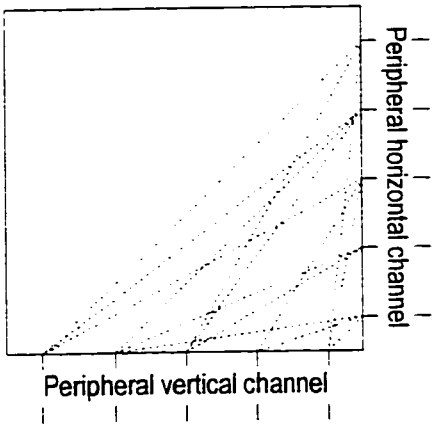


Figure 24: Peripheral corner SB in the top left corner.

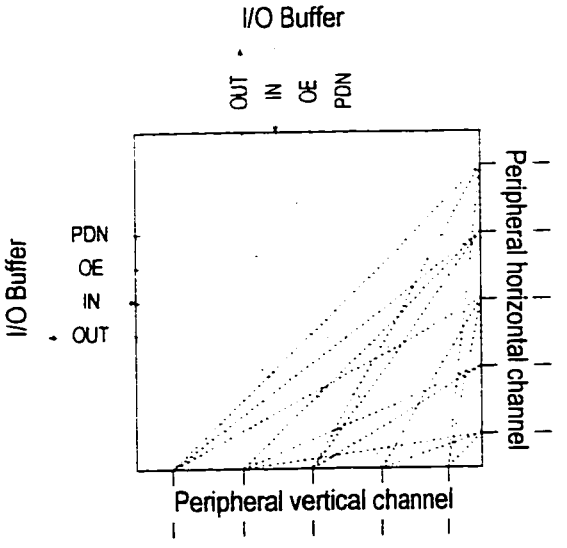


Figure 25: Peripheral corner SB connecting two I/O pads to the top and left peripheral channels.

2.2 Information Required by the Router

The router requires information regarding the design to be implemented as well as information about the FPGA platform to be used for implementation. Design information is given in the form of a netlist of connected UPMs. while FPGA information include the FPGA configuration and the routing structure parameters.

2.2.1 The FPGA Configuration

Information regarding the general FPGA array topology should be provided. Such information include:

- The number of FPGA columns and rows.
- The number of I/O pads on each side, and
- The UPM configuration i.e. single or back-to-back.

An integer is used to indicate if the UPM configuration is single or back-to-back. It is "0" if single UPM configuration is used and "1" if UPMs are back-to-back.

2.2.2 The Routing Structure

Details of all the routing resources in the FPGA should be described. These include:

- The pinout of UPMs,

- routing channels,
- connection boxes,
- switch boxes, and
- I/O pads.

All routing resources and their templates are described next.

2.2.2.1 The Pinout of Universal Programmable Modules

For single UPM configuration, description of one UPM template is enough. However, if the FPGA uses back-to-back configuration, two UPM templates are described: one description for the UPM on the left and one for the UPM on the right (refer to section 2.1.1, page 25 for details about the UPM pin types and numbers).

The UPM pinout is described by a separate line for each side in the following order: top, right, bottom, and left. Pins on the top side are numbered left to right, pins on right side are numbered top to bottom, pins on bottom side are numbered right to left, and pins on left side are numbered bottom to top (Figure 26). Each line describing a side has the following format:

P1 P2 P3 P4.....line-end

where

P_i a positive integer indicating the type of pin number i on that side

Line-end a character indicating the end-of-line (0).

After the description of the pinout of the four sides, the direction (input or output) of each pin must be indicated in two separate lines as follows:

Pin-use T1 T2 T3 T4.....line-end

where

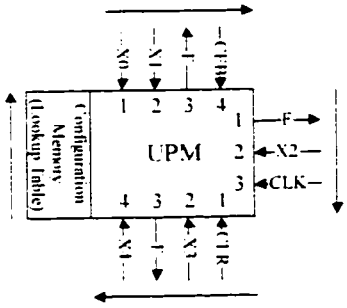
Pin-use a negative integer indicating the use of the pin: -1 indicates that the following pin types are used for output while -2 indicates that the following pin types are used for input

T_i a positive integer indicating the pin type T_i is used for *Pin-use*.

Line-end a character indicates end-of-line (0)

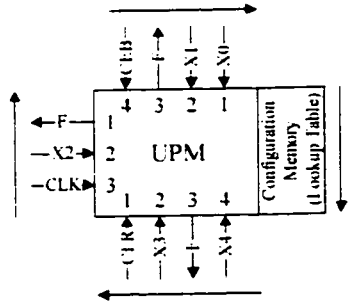
A *template-end* indicator should be given (0). Figure 26 shows a UPM and its pinout For back-to-back configuration, two UPM pinouts are described. The pinout of the UPM on the right is described as shown on Figure 26 and the one on the right is shown in Figure 27. The pin types are numbered as follows:

Pin type	X	F	CEB	CLK	CLR
Value	1	2	3	4	5




```
1 1 2 3 0      { pins on top }
2 1 4 0        { pins on right }
5 1 2 1 0      { pins on bottom }
0              { pins on left }
-2 1 3 4 5 0   { pins used as inputs}
-1 2           { pins used as outputs }
0              { template-end indicator }
```

Figure 26: A UPM and its pinout description.




```
3 2 1 1 0      { pins on top }
0              { pins on right }
1 2 1 5 0      { pins on bottom }
4 1 2 0        { pins on left }
-2 1 3 4 5 0   { pins used as inputs}
-1 2           { pins used as outputs }
0              { template-end indicator }
```

Figure 27: Left-UPM and its pinout description.

2.2.2.2 Routing Channels

The description of routing channels is done by giving the number of the tracks in the channel and the type of each track. The track type describes the types of segments used in this track, e.g. long line, even-doubly segmented, odd-doubly segmented, and fully segmented. Six different channel templates are needed:

1. Horizontal internal channel
2. Vertical internal channel
3. Left vertical peripheral channel
4. Right vertical peripheral channel
5. Top horizontal peripheral channel
6. Bottom horizontal peripheral channels

The information needed for each channel is included with the template description of the connection box on the channel.

2.2.2.3 Connection Boxes

Channels with different templates should have different connection box templates. Therefore, for single UPM FPGAs, six connection box (CB) templates are needed (two internal and four peripheral templates):

Internal

1. Vertical CB
2. Horizontal CB

Peripheral

1. CB on the left peripheral channel (vertical)
2. CB on the right peripheral channel (vertical)

3. CB on the top peripheral channel (horizontal)
4. CB on the bottom peripheral channel (horizontal)

Alternatively, for back-to-back configuration, two templates are required for the two connection box types used in the horizontal channels. Thus, nine connection box (CB) templates are needed (three internal and six peripheral CB templates):

Internal

1. Vertical CB
2. Left horizontal CB
3. Right horizontal CB

Peripheral

1. CB on the left peripheral channel (vertical)
2. CB on the right peripheral channel (vertical)
3. Left CB on the top peripheral channel (horizontal)
4. Right CB on the top peripheral channel (horizontal)
5. Left CB on the bottom peripheral channel (horizontal)
6. Right CB on the bottom peripheral channel (horizontal)

Each track passing through the connection box should be described by a separate line according to the following format:

$$\textit{Track-number}\ P_1\ P_2\ P_3\ P_4 \ldots 0\ Q_1\ Q_2\ Q_3 \ldots .\ \textit{Track-type}$$

where

Track-number the track number within the channel. The tracks are numbered top to bottom for horizontal connection boxes and left to right for vertical connection boxes.

P_i For vertical CBs: a positive integer that indicates a pin number on the right side of the UPM (or I/O pad) to the left of the track. The pin numbers are listed from the top to the bottom.

For horizontal CBs: a positive integer that indicates a pin number on the bottom of the UPM (or I/O pad) above the track. The pin numbers are listed from the right to the left.

0 Separator between the two sides

Q_i For vertical CBs: a positive integer that indicates a pin number on the left side of the UPM (or I/O pad) to the right of the track. The pin numbers are listed from the bottom to the top.

For horizontal CBs: a positive integer that indicates a pin number on the top of the UPM (or I/O pad) below the track. The pin numbers are listed from the left to the right.

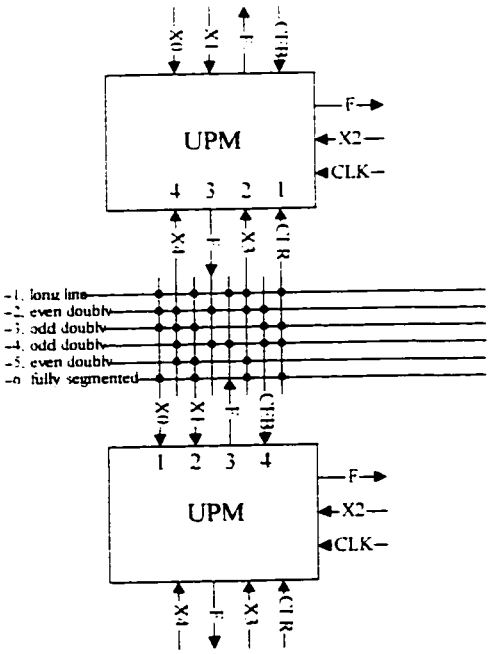
Track-type a negative integer used to indicate the track type as follows:

-1. Long line

-2. Even-doubly segmented

-3. Odd-doubly segmented

-4. Fully segmented.



1	1 2	0	1 2 3	-1
2	2 3 4	0	1 4	-2
3	1 4	0	1 2 4	-3
4	1 3 4	0	4	-3
5	2 4	0	3	-2
6	1 2	0	1 2 4	-4
0				

Figure 28: An internal horizontal connection box and its template description.

Note that, only pins that can be connected (via programmable switches) to the described track being listed in the P_i and Q_i fields. Each track description is terminated by a *template-end* indicator (0). Figure 28 shows an internal horizontal connection box and its template description.

2.2.2.4 Switch Boxes

Since FPGA routing channels have some doubly segmented tracks which may be either even or odd, the structure of a switch box depends on its location in the FPGA array (i.e. its row and column coordinates). Likewise, switch boxes on peripheral channels are different. Accordingly, sixteen switch box (SB) templates are needed: four internal SBs, eight peripheral SBs and four corner SBs.

Internal

1. SB at the intersection of even vertical and horizontal channels.
2. SB at the intersection of odd vertical and horizontal channels.
3. SB at the intersection of even vertical channel and odd horizontal channel.
4. SB at the intersection of odd vertical channel and even horizontal channel.

Peripheral

1. SB at the intersection of the leftmost vertical channel and an even horizontal channel.
2. SB at the intersection of the leftmost vertical channel and an odd horizontal channel.
3. SB at the intersection of the rightmost vertical channel and an even horizontal channel.

4. SB at the intersection of the rightmost vertical channel and an odd horizontal channel.
5. SB at the intersection of an even vertical channel and the topmost horizontal channel.
6. SB at the intersection of an odd vertical channel and the topmost horizontal channel.
7. SB at the intersection of an even vertical channel and the bottommost horizontal channel.
8. SB at the intersection of an odd vertical channel and the bottommost horizontal channel.

Corner

1. SB at the top left corner.
2. SB at the top right corner.
3. SB at the bottom left corner.
4. SB at the bottom right corner.

Channel tracks connected to a switch box are numbered in a clockwise direction starting from the top side leftmost track continuing on the right side, the bottom side, then the left side. All tracks are numbered including the tracks that pass through the SB without interruption (Figure 29). The peripheral (corner) SBs have one (two) side(s) which is (are) not connected to channel tracks. In this case, the number of pins on this side is assumed equal to the number of tracks on the channel on the opposite side. Therefore, the number of the topmost track on the left side of the SB will be equal to 2*NVch+2*NHch, where

NVch number of vertical channel tracks,

NHch number of horizontal channel tracks.

Pins on the top, right and bottom sides[1] of the switch box are described each in separate lines according to the following format:

Pin-number P_1 P_2 P_3 P_4 ... *Line-end*

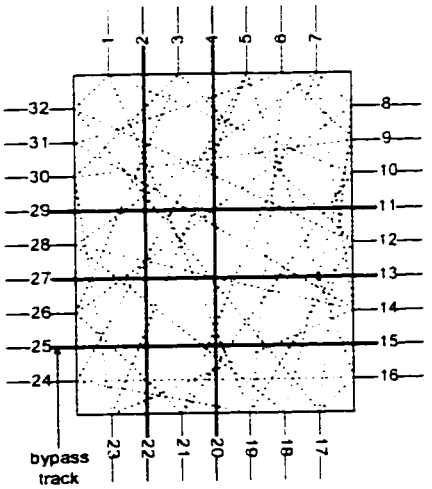
where

Pin-number a positive integer represents the number of the pin to be described.

P_i a positive integer grater than[2] *Pin-number* indicates that the switch box can connect the track segment connected to pin number P_i to the track segment connected to pin number *pin-number.*

Line-end line end character (0).

A *template-end* indicator (0) is also used. Figure 29 shows an internal switch box and its template description.




```
1    10 19 32 0
3    12 26 30 0
5    16 23 24 0
6    9  19 31 0
7    8  14 21 28 0
8    17 26 0
9    18 30 0
10   21 0
12   19 32 0
14   23 28 31 0
16   17 24 0
17   32 0
18   28 0
19   24 0
21   26 0
23   30 0
0
```

Figure 29: An internal switch box and its template description.

Peripheral and corner SBs can be used to connect I/O pad pins to the peripheral channel. In this case, the first I/O pad pin is associated to the first pin on the SB side

[1] switches connected to the pins on the left side will be already described by the other three sides
[2] switches connected to the pins "less than" Pin-number will be already described by the previous lines

facing the I/O pad. The second I/O pad pin is associated with the second the third with the third and so on. Figure 30 shows a top left corner switch box used to connect two I/O pads to peripheral channels together with its template description.

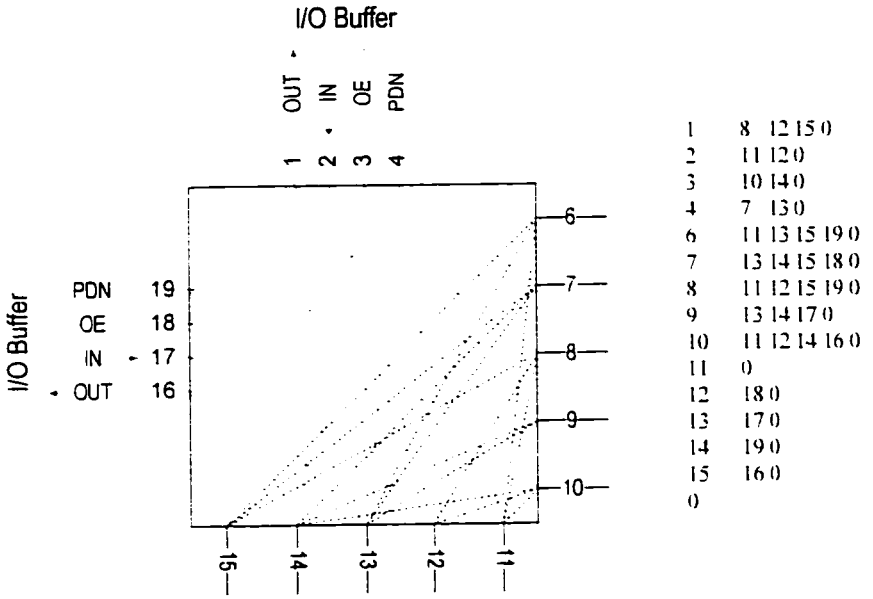


Figure 30: Top-left Corner Switch box connecting I/O pads to peripheral channels and its template.

2.2.2.5 I/O Pads

Since the I/O pads on all sides have the same structure, only one I/O pad template is required. The side on which the pad is located determines the side on which its pins are made available. The I/O pad template is described by one line with the following format:

$$P_1\ P_2\ P_3\ P_4 \ldots\ldots\ldots\ldots \textit{line-end}$$

where

P_i a negative integer indicating the type of pin number i.

Line-end a character indicates end-of-line (0)

Figure 31 shows an I/O pad on the top FPGA side and its template description where

Pin type	OUT	IN	OE	PDN
Value	-1	-2	-3	-4

The pin types are listed from left to right.

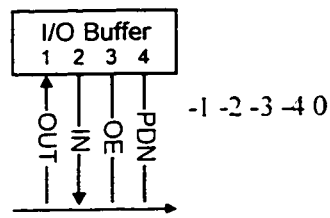


Figure 31: An I/O pad and its template description.

2.2.3 FPGA Template File Format

The FPGA template file lists FPGA configuration and its routing structure in the following order:

- The number of rows
- The number of columns
- single or back-to-back UPMs
- number of pads on top
- number of pads on right
- number of pads on bottom
- number of pads on left

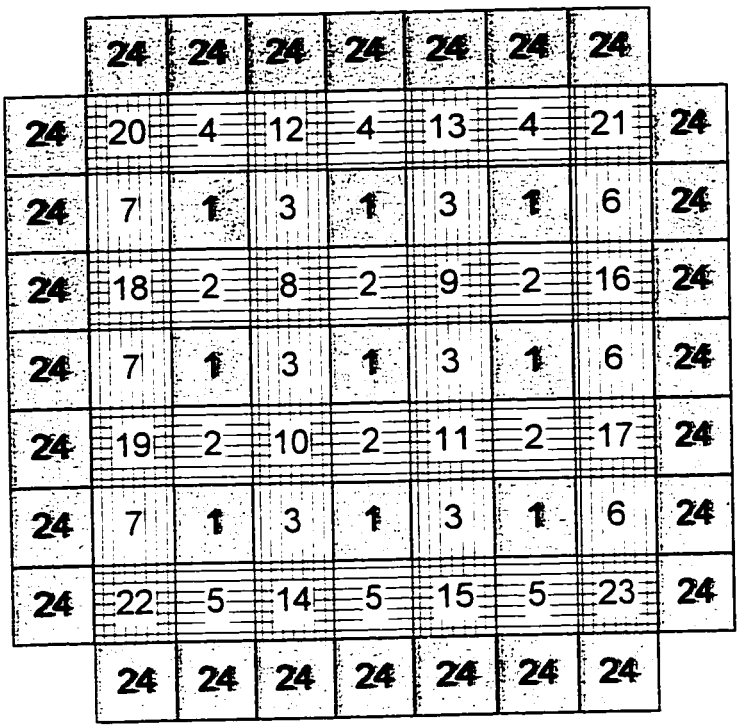
The description of the routing structure for single FPGAs follows in the following order:

1. UPM pinout
2. Internal horizontal CB
3. Internal vertical CB
4. Peripheral horizontal CB in the top peripheral channel
5. Peripheral horizontal CB in the bottom peripheral channel
6. Peripheral vertical CB in the right peripheral channel
7. Peripheral vertical CB in the left peripheral channel
8. Internal Even-Even SB
9. Internal Even-Odd SB
10. Internal Odd-Even SB
11. Internal Odd-Odd SB
12. Peripheral Even SB on the top peripheral channel
13. Peripheral Odd SB on the top peripheral channel
14. Peripheral Even SB on the bottom peripheral channel
15. Peripheral Odd SB on the bottom peripheral channel
16. Peripheral Even SB on the right peripheral channel
17. Peripheral Odd SB on the right peripheral channel
18. Peripheral Even SB on the left peripheral channel
19. Peripheral Odd SB on the left peripheral channel
20. Corner SB on top-left
21. Corner SB on top-right
22. Corner SB on bottom-left
23. Corner SB on bottom-right.
24. I/O pad template description.

Alternatively, description of the routing structure of back-to-back FPGAs is given as:

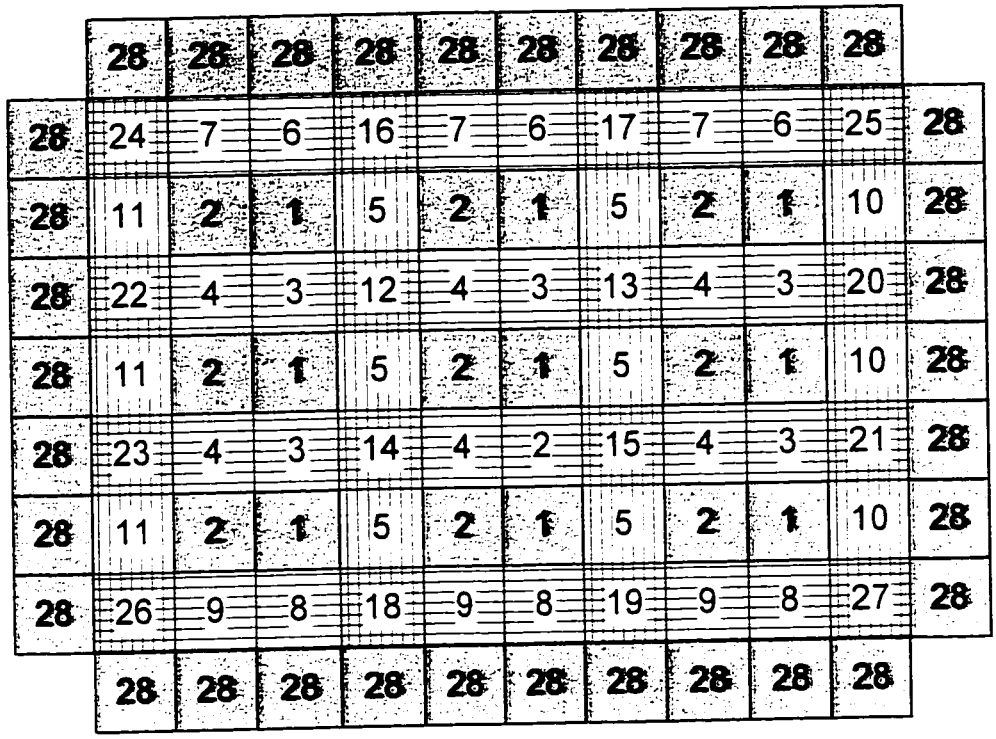
1. Right UPM pinout
2. Left UPM pinout
3. Internal right horizontal CB
4. Internal left horizontal CB
5. Internal vertical CB
6. Right peripheral horizontal CB in the top peripheral channel
7. Left peripheral horizontal CB in the top peripheral channel
8. Right peripheral horizontal CB in the bottom peripheral channel
9. Left peripheral horizontal CB in the bottom peripheral channel
10. Right peripheral vertical CB in the peripheral channel
11. Left peripheral vertical CB in the peripheral channel
12. Internal Even-Even SB
13. Internal Even-Odd SB
14. Internal Odd-Even SB
15. Internal Odd-Odd SB
16. Peripheral Even SB on the top peripheral channel
17. Peripheral Odd SB on the top peripheral channel
18. Peripheral Even SB on the bottom peripheral channel
19. Peripheral Odd SB on the bottom peripheral channel
20. Peripheral Even SB on the right peripheral channel
21. Peripheral Odd SB on the right peripheral channel
22. Peripheral Even SB on the left peripheral channel
23. Peripheral Odd SB on the left peripheral channel
24. Corner SB on top-left
25. Corner SB on top-right
26. Corner SB on bottom-left
27. Corner SB on bottom-right.
28. I/O pad template description.

Figure 32 and Figure 33 show templates for the single and back-to-back UPMs configurations respectively.



1. UPM
2. Internal horizontal CB
3. Internal vertical CB
4. Peripheral horizontal CB in the top peripheral channel
5. Peripheral horizontal CB in the bottom peripheral channel
6. Peripheral vertical CB in the right penpheral channel
7. Peripheral vertical CB in the left peripheral channel al channel
8. Internal Even-Even SB
9. Internal Even-Odd SB
10. Internal Odd-Even SB
11. Internal Odd-Odd SB
12. Peripheral Even SB on the top peripheral channel
13. Peripheral Odd SB on the top peripheral channel
14. Peripheral Even SB on the bottom peripheral channel
15. Peripheral Odd SB on the bottom peripheral channel
16. Peripheral Even SB on the right peripheral channel
17. Peripheral Odd SB on the right peripheral channel
18. Peripheral Even SB on the left peripheral channel
19. Peripheral Odd SB on the left peripheral channel
20. Corner Left-Top SB
21. Corner Right-Top SB
22. Corner Left-Bottom SB
23. Corner Right-Bottom SB
24. I/O pad .

Figure 32: The 24 different templates in the single UPM FPGA configuration.



1. *Right* UPM
2. *Left* UPM
3. *Right* Internal horizontal CB
4. *Left* Internal horizontal CB
5. Internal vertical CB
6. *Right* Peripheral horizontal CB in the top peripheral channel
7. *Left* Peripheral horizontal CB in the top peripheral channel
8. *Right* Peripheral horizontal CB in the bottom peripheral channel
9. *Left* Peripheral horizontal CB in the bottom peripheral channel
10. Peripheral vertical CB in the right peripheral channel
11. Peripheral vertical CB in the left peripheral channel al channel
12. Internal Even-Even SB
13. Internal Even-Odd SB
14. Internal Odd-Even SB
15. Internal Odd-Odd SB
16. Peripheral Even SB on the top peripheral channel
17. Peripheral Odd SB on the top peripheral channel
18. Peripheral Even SB on the bottom peripheral channel
19. Peripheral Odd SB on the bottom peripheral channel
20. Peripheral Even SB on the right peripheral channel
21. Peripheral Odd SB on the right peripheral channel
22. Peripheral Even SB on the left peripheral channel
23. Peripheral Odd SB on the left peripheral channel
24. Corner Left-Top SB
25. Corner Right-Top SB
26. Corner Left-Bottom SB
27. Corner Right-Bottom SB
28. I/O pad .

Figure 33: The 28 different templates in the back-to-back FPGA configuration.

2.2.4 Netlist File Format

The netlist of the design to be implemented should be provided to the router in a separate file. Next, the adopted netlist file format that will be read by the router is described. Each net is described in a separate line using the following format:

net-name x_1 y_1 z_1 x_2 y_2 z_2 ..... *net-type* *pad-num*

Where

net-name String of characters representing the net name.

x_i, y_i positive integers representing UPM row and column coordinates respectively (Figure 34). In the back-to-back UPM structure (Figure 35), the position of the UPM (left or right) will be known from its column coordinate. UPMs on columns 2, 5, 8, 11, 14..... are left UPMs, while UPMs on columns 3, 6, 9, 12, 15..... are right UPMs.

Z_i positive integer indicating pin type. The following table shows a possible set of pin types. Other pin types may be added.

Pin type	X	F	CEB	CLK	CLR
Value	1	2	3	4	5

net-type non-positive integer representing the net type:

net type	value
internal net	0
net is connected to primary output	-1
net is connected to primary input	-2
net is connected to both primary input and output	-3

pad-num a positive integer representing the pad number the net is to be connected to. I/O pads will be numbered starting from the left most pad on the upper FPGA side with 1. The numbering is clockwise incremented by 1 (Figure 36). If *pad-num* is 0 it means that there is no specific pad to be used. If so, the router will find the closest available pad to net pins. If the net is internal this field is not needed.

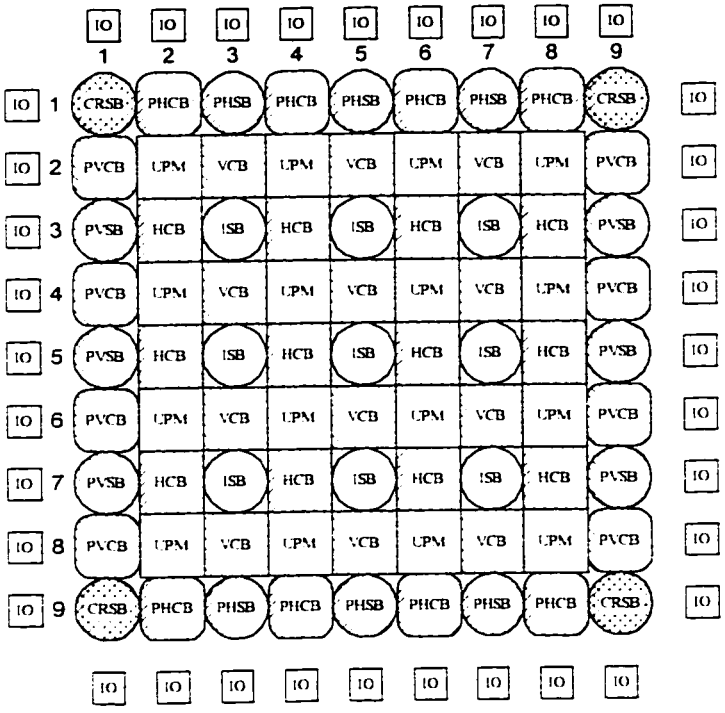


Figure 34: Numbering rows and columns in single UPM configuration.

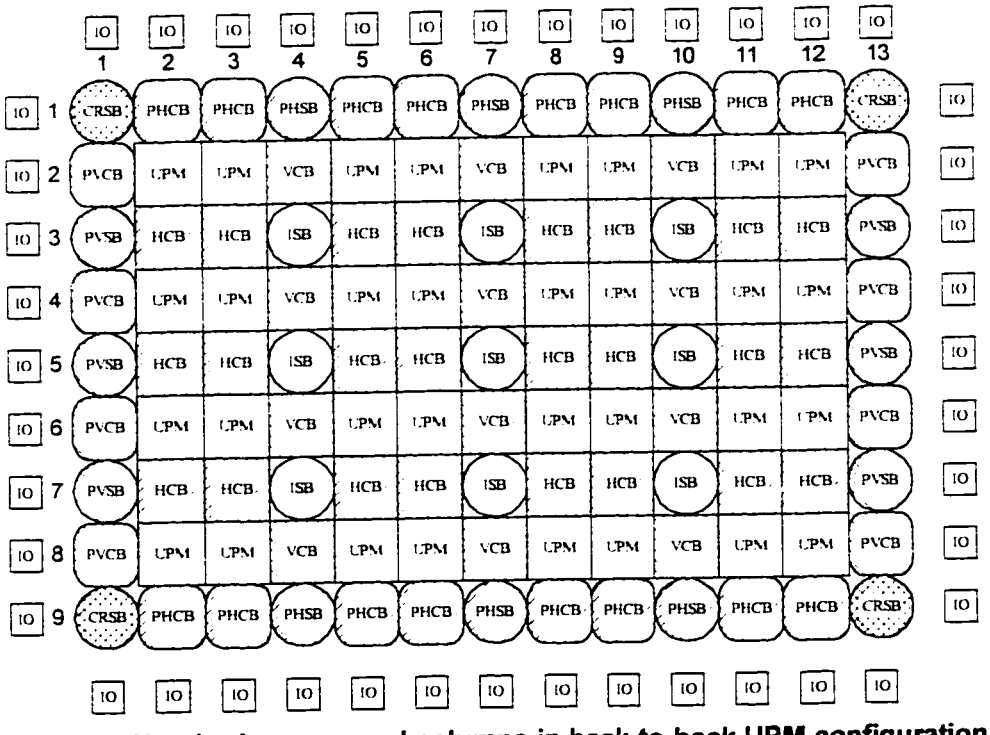


Figure 35: Numbering rows and columns in back-to-back UPM configuration.

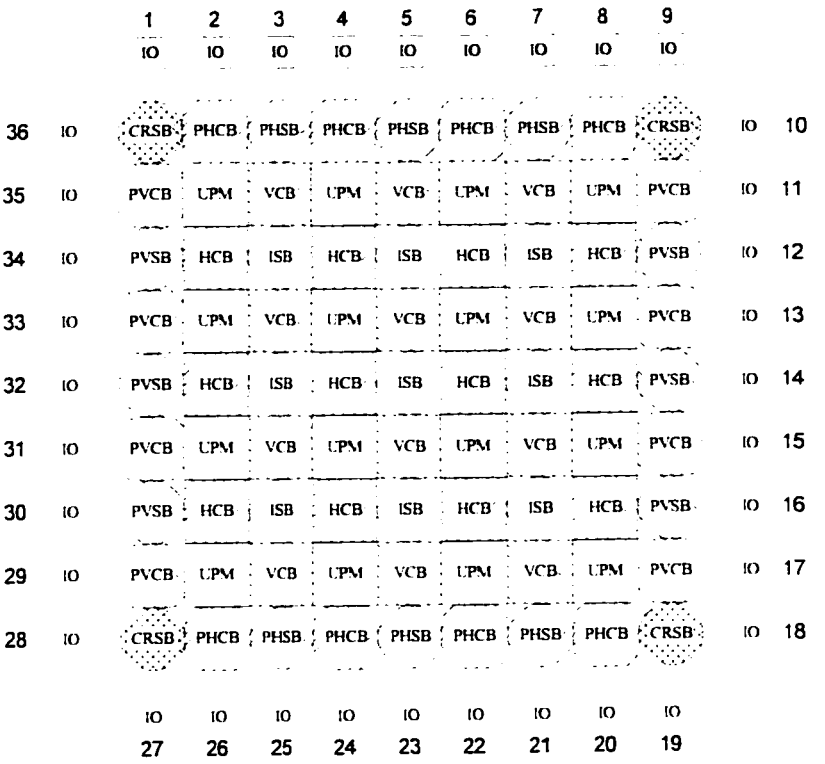


Figure 36: Numbering I/O pads.

Since the router is order dependent, critical nets should be listed first. In addition, the nets that have pre-specified I/O pad locations should be placed before the nets with unspecified I/O pad locations. Nets are assigned one of 6 possible priority. These levels are listed in a descending order as follows:

1. Critical net having an I/O pad with pre-specified location.
2. Critical net having an I/O pad with floating or unspecified location.
3. Critical net with no primary I/O (no I/O pads in the net).
4. Non-critical net having an I/O pad with pre-specified location.
5. Non-critical net having an I/O pad with floating or unspecified location.
6. Non-critical net with no primary I/O (no I/O pads in the net).

The file is terminated by the keyword "**END.**" in the *net-name* field.

CHAPTER 3

GLOBAL ROUTER

The routing problem is performed in two steps: global routing followed by detailed routing. Global routing performs several tasks. It estimates whether the netlist is routable within the given FPGA. Furthermore, the global router finds the shortest possible Steiner tree for each net and check if this tree satisfies the time constraints. Moreover, it provides the detailed router with the FPGA array blocks that are to be used for routing each net.

The global router reads the general array layout, routing resources architecture and the design netlist (section 2.2, page 34). The global router does not need all the details of the routing resources; therefore, it will only extract the information it requires from the input templates. The global router requires

- For UPMs, the number of pins of each pin-type available on each side.
- For CBs, the number of available tracks.
- For SBs,
  - The number of pins available on each side, and

- The number of available switches for the six possible connections (Figure 9, page 19). This includes number of available switches from
  - the left side to the right side,
  - the left side to the top side,
  - the left side to the bottom side,
  - the right side to the top side
  - the right side to the bottom side
  - the top side to the bottom side.

  The number of bypass tracks is added to the corresponding field. If T tracks bypass the switch box horizontally, the number of switches from right to left and from left to right is incremented by T.
- For I/O pads, the number of I/O pads per peripheral CB or SB.

The global router produces the following for each net:

- The maximum number of blocks passed to connect one of the sinks to the source pin.
- A solution tree through the FPGA blocks.

The proposed solution for the global router uses a variation of Lee maze routing algorithm (see [18] for details). An overview of Lee algorithm is given next. Follows the description of the global routing algorithm. The algorithm is illustrated next with an example. Finally, the algorithm is analyzed.

3.1 Lee Algorithm

Lee algorithm is the most widely known maze routing algorithm. When the algorithm is applied to find a path between two points on a grid that has obstacles, it is guaranteed that it will find the shortest available path [18]. The algorithm has three phases: (1) filling phase, (2) retracing phase, and (3) label clearance phase. Lee algorithm is generally used to route two-terminal nets, however, it can be used to route multi-terminal nets [18].

In the filling phase, considering two-pin net, the algorithm labels one of the two pins as a source S, and the other as a target T. Filling is done in an expanding diamond fashion. The filling phase starts with filling all free cells next to S with 1. Then, all cells that are adjacent to each cell with label 1 are labeled with 2. The process goes on, each free cell adjacent to a cell with value k is labeled with k+1 (Figure 37). In the k^{th} filling step, all non-blocking cells with Manhattan distance of k from S are given a label k. Filling stops when at step k one of the following happens:

- the target T is reached.
- there is no cell to be labeled with k (dead end), or
- k is greater than a maximum allowed path length MaxL.

If the target is reached with a label less than the pre-specified limit (MaxL) then Lee algorithm will proceed to the retrace phase, otherwise, it will report "no path could be found with length less than MaxL". If target T is reached in step k, the path from

source S to target T is of length k.

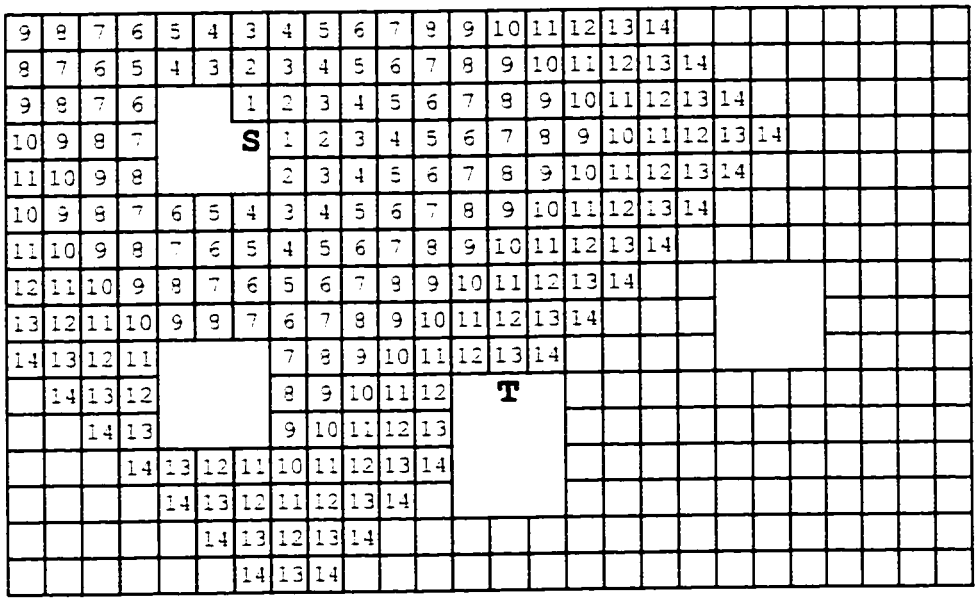


Figure 37: The filling phase in Lee algorithm.

The second phase, retracing, is the opposite of the filling phase. During this phase, for a target **T** found in step k, the algorithm looks for an adjacent cell to **T** with label k-1. Then, it looks for a cell adjacent to that labeled k-1 with label k-2. If there are more than one cell found with label k-2, the cell that results in lesser number of turns is chosen. This process goes on until the source **S** is reached. Figure 38 shows retracing the path from the target found in step 14 of the filling phase (filling is shown in Figure 37). The shaded blocks on Figure 38 represent the path found.

After retracing, the labeled cells should be cleared in the last phase of Lee algorithm. In this phase, all cells labeled in phase 1 are cleared except the cells that belong to the path found in phase 2, which will be marked as obstacles.

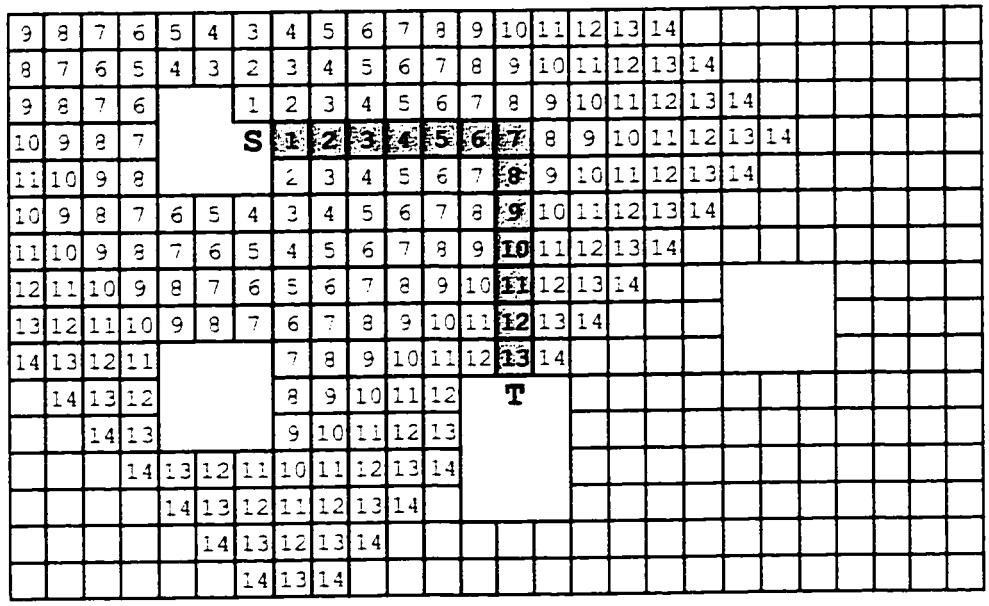


Figure 38: The retracing phase in Lee algorithm.

3.2 Global Routing Algorithm

A 2-D array is used to model the FPGA. All connection boxes and switch boxes in the FPGA are represented as empty cells while all UPMs and I/O pads other than those belonging to the net are represented as obstacles (Figure 39). UPMs and I/O pads belonging to the net are labeled either as sources or as targets. Rows and columns of the FPGA grid are numbered as shown in Figure 39. The global routing algorithm outline is given in Figure 40.

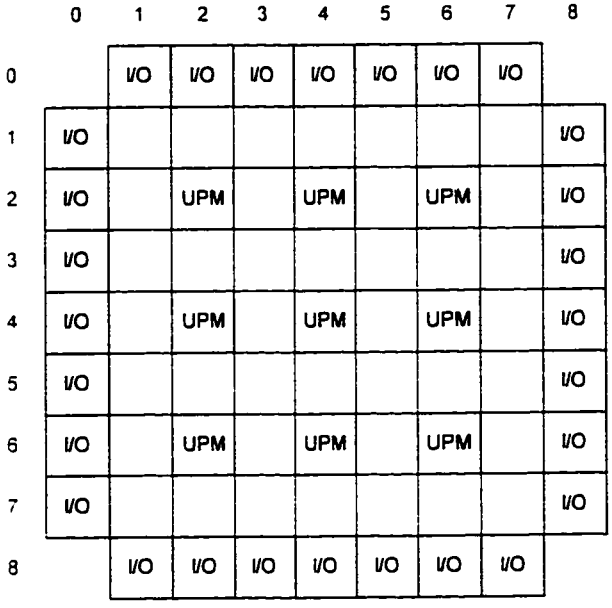


Figure 39: Modeling a 7*7 FPGA with single UPM configuration.

```
Procedure Global_router
Begin
   For each net do
      1. Specify source(s) and targets(s)
      2. Do until all targets are reached
         • Propagate one step
         • If target is found during propagation do
            • Retrace to find all paths to a source
            • Store path(s)
      3. Find cost of all paths stored in 2
      4. Add path with lowest cost to solution-tree
      5. Clear all labels
      6. Update routing resources
      7. Consider solution-tree blocks as sources and the yet-unconnected
         nodes in the net as targets
      8. Go back to step 2 until all nodes of the net are connected
      9. Report solution-tree for the current net
   End for
End
```

Figure 40: Global routing algorithm.

3.2.1 Source(s) and Target(s)

The first step in the algorithm is to specify sources and targets. Initially, for a new

net, the source is chosen from the net pins. However, as the algorithm starts building solution tree, the sources are chosen from the solution tree blocks.

3.2.1.1 Initial Source

The source to be chosen should be the driver of the net. Thus, each UPM that has an output pin in the net is labeled as a source. If the net is connected to a primary input, all free I/O pads are labeled as sources unless the position of the I/O pad that the net should be connected to is pre-specified. In this case, only the pre-specified I/O pad is labeled as source and the rest as obstacles.

All unlabeled net UPMs will be labeled as targets. If the net is connected to a primary output, all free I/O pads are considered targets unless the position of the I/O pad that the net should be connected to is specified. In this case, the specified I/O pad is labeled as target and the rest as obstacles.

3.2.1.2 Solution Tree Blocks

In step number 7 in the algorithm (Figure 40), source(s) is(are) chosen from the blocks on the solution tree under-construction. This would help in finding a short Steiner tree that connects the net pins. Since the SBs and CBs in use can connect to more than one direction simultaneously, all SBs, and CBs in the solution tree are considered sources. The SBs and CBs can be used as Steiner points.

All UPMs on the solution tree with input pins on the net are not marked as sources since one of their input pins is already used. However, a UPM on the solution tree with an output pin in the net is labeled as source if its output is available on more than one side. An I/O pad in the current solution tree is not labeled as source since an I/O pad can be used for one net only. The yet unconnected net pins are labeled as targets as shown previously.

3.2.2 Propagation

The second step in the algorithm is label propagation. The label propagation procedure is shown in Figure 41. The source(s) is(are) is labeled first. All UPMs and I/O pads are labeled with zeros. Each SB and CB is labeled with a value indicating the number of blocks passed to reach this block from the initial source.

```
Procedure Propagate
Begin
   Label all blocks considered as sources with initial values
   Push all sources blocks into the current-blocks-stack
   LOOP: until found or current iteration is greater than the maximum
      For each block in current-blocks-stack do
         For all the blocks around the current-block do
            If the block is reachable from the current-block then
               If the block is target then found is true
               Else if the block is free then
                  Label block with current-block-label+1
                  Push block into the new-blocks-stack
               End-if
            End-if
         End-for
      End-For
      If new-block-stack is empty then
         Exit LOOP and return "DEAD-END"
      Else current-blocks-stack := new-blocks-stack
   End-LOOP
END
```

Figure 41: Propagation procedure.

3.2.3 Reachability

Block B is said to be reachable from an adjacent block A if there is at least one available connection from A to B. Reachability from one block (B) to another (A) depends on the type of the two blocks. Figure 42 shows how reachability is determined. Reachability is only considered between two adjacent blocks. The adjacency side is that side of block A which is adjacent to one of block B sides.

```
Function Reachable(A,B)
Begin
   Reachable := False
   Case A of
   IPM:              {B must be a CB}
      if A has available pin of the required type on the adjacency side
      and B has available free track then Reachable:= True
   CB:               {B is either IPM, SB or I/O pad}
      Case B of
      IPM or I/O pad:
         if B has available pin of the required type on the adjacency side
         and B is a source or target then Reachable:= True
      SB:
         if B has available pin on the adjacency side then Reachable:= True
      End case
   SB:               {B is either a CB or an I/O pad}
      Case B of
      CB:
         if A has available pin on the adjacency side and
         B has available free track then Reachable:= True
      I/O pad:
         if B has available pin of the required type on the adjacency side
         and B is a source or target then Reachable := True
      End case
   I/O pad:          {B is either CB or SB}
      Case B of
      CB:
         If A has available pin of the required type on the adjacency side
         and B has available free track then Reachable:= True
      SB:
         If A has available pin of the required type on the adjacency side
         and B has available pin on the adjacency side then Reachable:= True
      END case
   END case
End
```

Figure 42: Reachability function.

3.2.4 Retracing

If the propagation step reports a found target(s), the algorithm will start retracing step to find a segment that can be a branch on the solution tree. Starting from the target, it should find all possible paths to one of the source blocks. This is done using the recursive procedure shown in Figure 43. Note that this procedure will find all possible paths from the target to one of the sources.

```
Procedure RETRACE (CRNT-BLOCK)
Begin
   For all BLOCKS around the CRNT-BLOCK do
      If BLOCK is reachable from the CRNT-BLOCK then
         If BLOCK label < CRNT-BLOCK label then
            Store CRNT-BLOCK
            CALL RETRACE(BLOCK)
         Else if BLOCK is a source then
            Store BLOCK
            REPORT PATH
         End-if
      End-if
   End-for
END
```

Figure 43: Retracing procedure.

3.2.5 Cost Function

The cost of a given path is defined as a function of three parameters: the path congestion, average distance from other nodes, and path delay. The cost function will take the three parameters and decide which path has lower cost. Fuzzy logic is used to evaluate the goodness of any particular path. A similar fuzzification approach to that presented in [4] is used. Next, an overview on applying fuzzy logic to routing is given together with description of each of the three cost parameters.

3.2.6 Fuzzy Logic for Routing

Balancing different objectives by weight functions is difficult because it requires comparison of "apples" and "oranges". Fuzzy logic is a convenient vehicle for solving this problem. It allows mapping values of different criteria into linguistic values. The linguistic values characterize the level of satisfaction with the objectives numerical values. Fuzzy logic operates over values in the interval [0,1] defined by the membership functions for each objective [4]. During routing of a particular net, one has to evaluate different routes based on several conflicting objectives. Therefore, the objective function is generally a vector-function

$$\bar{F}(x) = (f_1(x), f_2(x), \ldots\ldots, f_n(x))$$

where $f_i(x)$ is the value of the i^{th} objective, $1 \le i \le n$. To obtain a fuzzy logic definition of the above multicriteria objective function, n linguistic variables are defined for these functions. To simplify the implementation, only one linguistic value is defined for each variable. These linguistic values characterize the degree of satisfaction with the values of various objectives $f_i(x)$, $1 \le i \le n$. These degrees of satisfaction are described by membership functions $\mu_i(.)$ on fuzzy sets of linguistic values.

The most desirable routing alternative is the one with the highest membership in the fuzzy subsets *low delay, low congestion, and short distance*. However, such a solution most likely does not exist since objective criteria generally conflict with one another. Therefore, one has to tradeoff individual criteria against each other. Such

tradeoff is conveniently specified in linguistic terms in the form of one or several fuzzy logic rules. A fuzzy logic rule is an **If-Then** rule. The **If** part (*antecedent*) is a fuzzy predicate defined in terms of linguistic values and fuzzy operators (**AND** and **OR**). The **Then** part is called the *consequent*. In our case, the linguistic value used in the consequent part identifies the fuzzy subset of good solutions. Therefore, the result of evaluation of the antecedent part identifies the degree of membership in the fuzzy subset of good routing solutions according to the fuzzy rule in question. Looking for net paths with *low delay*, *low congestion*, and *short distance*, fuzzy c rules would be:

- **R.1**: **If** p has *low congestion* **Then** p is a good path
- **R.2**: **If** p has *short distance* **Then** p is a good path
- **R.3**: **If** p *has low delay* **Then** p is a good path

The **If**-part of each rule consists of a single antecedent, and the **Then**-part is the consequent, which is the same for all three rules and is the fuzzy subset *good solution*. The **Then**-part of rule **R.i** assigns a degree of membership in the fuzzy subset of *good paths* with respect to objective i, i = 1,2,3. For example, rule **R.1** defines the membership degree of path p in the fuzzy subset *of low congested paths* over the universe of all possible paths for a particular net. To combine the outcome from the three rules, the *additive combiner operator* [4] illustrated in Figure 44 can be used. According to this operator, the membership of a particular solution p in the good solutions fuzzy subset with respect to the three rules is computed as follows:

$$\mu_{(S)}(p) = w_1 * \mu_{(S \circ R.1)} + w_2 * \mu_{(S \circ R.2)} + w_3 * \mu_{(S \circ R.3)}$$

where w_i is the degree of belief in Rule **R.i**, i=1,2,3, and S stands for *good path*.

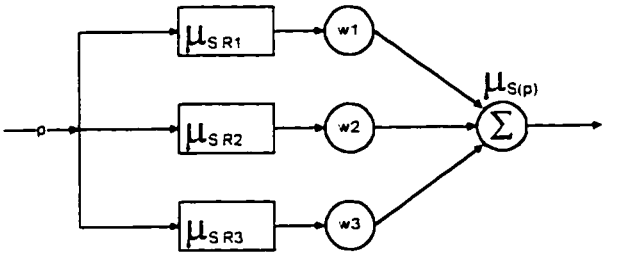


Figure 44: The additive combiner fuzzy operator.

Another way to combine the outcome from the three rules is the *Ordered Weighted Average* (OWA) operator [4] illustrated in Figure 45. According to this operator, the membership of a particular solution p in the fuzzy subset of good solutions with respect to the three rules is computed as follows:

$$\mu_{(S)}(p) = \beta * MAX(\mu_{(S^{\circ}R.1)}, \mu_{(S^{\circ}R.2)}, \mu_{(S^{\circ}R.3)}) + (1-\beta) * AVG(\mu_{(S^{\circ}R.1)}, \mu_{(S^{\circ}R.2)}, \mu_{(S^{\circ}R.3)})$$

where β is a constant in the interval [0,1], MAX will supply the maximum membership function, AVG will find the average of the three membership functions, and S stands for *good path.*

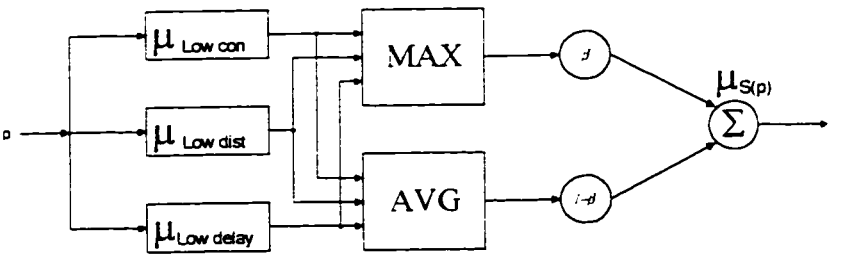


Figure 45: The Ordered Weighted Average fuzzy combiner.

The optimal solution is found as

$$F(p) = max\ \mu_{(S)}(p).$$

The fuzzification approach given above exhausts neither all possible factors to be considered in routing nor their dependencies, but it demonstrates how a traditional definition of a multiobjective problem can be transformed into a fuzzy logic definition.

3.2.6.1 Congestion

Path congestion is defined to be the congestion of the most congested block in that path: $CON(path) = \underset{\forall \text{ blocks } \in \text{ path}}{MAX}(BC)$ and $BC = U/A$ where

BC stands for Block Congestion.
U is the Used resources, and
A is the number of initially Available resources.

U of a block is the number of paths that traverse this block using a switch or a track in this block that can be used to perform the required connection. A is the total number of switches or tracks in this block that can be used to perform the required connection. If a path is to use a switch box to connect top to left, U is the number of already routed paths which use this SB to connect from top to left. A is the total number of switches in this SB from top to left. However, if a path is to use a connection box to connect from top to left, U is the number of the already routed paths which use this CB, while A is the total number of tracks passing through it.

The membership function $\mu_{Low\_con}(CON(p))$ in the fuzzy subset of low congested

paths is shown in Figure 46. It has been shown that typical routing resources utilization is about 80% [18]. Therefore, the congestion should not exceed 0.8.

$$\mu_{Low\_con}(x) = \begin{cases} 1 - \frac{2}{7}x & 0 \le x \le 0.7 \\ \frac{32}{5} - 8x & 0.7 \le x \le 0.8 \\ 0 & 0.8 \le x \le 1 \end{cases}$$

where $X = CON(Path)$

Figure 46: Low congestion membership function.

3.2.6.2 Distance to Unconnected Nodes

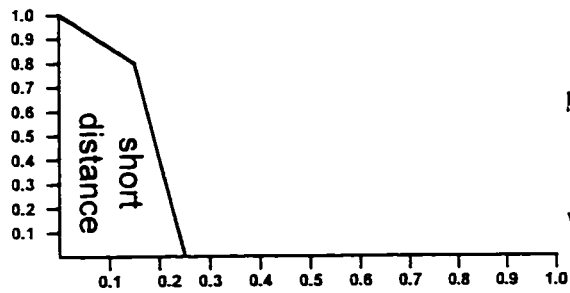
The distance to the yet unconnected nodes is the sum of Manhattan distances between all the blocks on the path and the yet unconnected net pins averaged over NB, the total number of blocks in the path. The source and target blocks are not considered path blocks.

$$DIST(path) = \frac{1}{NB} \sum_{\substack{\forall \text{ blocks} \\ b \in path}} \left[\sum_{\substack{\forall \text{ unconnected pins} \\ n \in net}} (|x_b - x_n| + |y_b - y_n|) \right]$$

This quantity is normalized to a value between 0 and 1 by dividing it by the upper bound value $Dist_{Max} = (N_{rows} + N_{colm}) * (N_{nodes} - 2)$ where N_{rows}, N_{colm} and N_{nodes} are number of rows, columns and net nodes respectively.

A normalized short distance membership function μ_{short_dist} is shown in Figure 47.

This function assumes that all path blocks fall within a bounding box for all net pins, and that the bounding box does not exceed 25% of $Dist_{Max}$.



$$\mu_{Short\_dist}(x) = \begin{cases} 1 - \frac{4}{3}x & 0 \le x \le 0.15 \\ 2 - 8x & 0.15 \le x \le 0.25 \\ 0 & 0.25 \le x \le 1 \end{cases}$$

where $x = norm(DIST(path))$

Figure 47: Short distance membership function.

3.2.6.3 Path Delay

Path delay (PD) is a function of NB, the number of blocks the path passes through, and NT, the number of switch boxes the path makes turns on. PD is estimated as

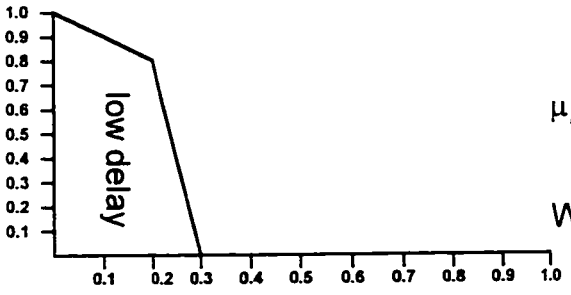
$$PD = \alpha * NB + \beta * NT$$

where α and β represent delay estimates of a unit wire segment and a switch respectively. PD is normalized between 0 and 1 as follows:

$$norm(PD) = \frac{PD}{(\alpha + \beta)(R + C)}$$

where (R + C) represent the maximum number of wire segments and switches needed to route a 2-pin net.

Figure 48 shows a membership function μ_{Low_delay} of the fuzzy subset of low delay paths. The Figure assumes that the delay for a given design should not exceed 30% of the normalizing factor $(\alpha + \beta)(R + C)$.



$$\mu_{Low\_delay}(x) = \begin{cases} 1-x & 0 \le x \le 0.2 \\ \frac{12}{5} - 8x & 0.2 \le x \le 0.3 \\ 0 & 0.3 \le x \le 1 \end{cases}$$

Where $x = norm(PD(path))$

Figure 48: *Low delay* membership function.

3.2.6.4 Total Cost Membership Function

The fuzzy cost of a path p is set equal to the value of the membership μ_{Low_Cost} of p in the fuzzy subset of low-cost paths with respect to the above three criteria.

$$\mu_{Low\_Cost} = w1 * \mu_{Low\_con} + w2 * \mu_{Short\_dist} + w3 * \mu_{Low\_delay}$$

where w1+w2+w3 = 1. The best path is the path with the highest μ_{Low_Cost}.

3.2.7 Building Solution Tree

After the retracing step has reported the path with the lowest cost, the algorithm starts/continues building the solution Steiner tree. If the found path is the first one for the net, it is considered as the first branch in the tree. Otherwise, this path is added to the solution tree as a new branch.

3.2.8 Label Clearance

After a path is selected, all the labeled blocks during the propagation step are cleared. However, if the net is to be connected to primary input or output and a path is found to one of the I/O pads, all other free I/O pads are marked as obstacles. The I/O pads marked as obstacles will remain so during the following steps until the completion of the current net solution. When the algorithm starts finding solution for a new net, the unused I/O pads are cleared (marked free).

3.2.9 Updating Blocks Congestion

The global router in use is an order dependent router. The routing resources are updated every time a net (or one of its segments) is routed. The resources are updated in the following manner:

- UPM: decrement number of available pins of the used pin-type on the used side.
- CB: decrement number of available tracks on the used CB.
- SB: if the switch box is used from side1 to side2, decrement number of available switches from side1 to side2. Number of available pins on both sides (side1 and side2) is decremented too.
- I/O pads: the used I/O is marked used (not free).

3.3 Example

In this section, the global routing algorithm is illustrated with an example. Consider the FPGA in Figure 49. A net to be routed consists of three pins, the output F of the $UPM_{2,3}$ (2^{nd} row and 3^{rd} column), an input to the $UPM_{2,9}$, and an input to the $UPM_{6,5}$.

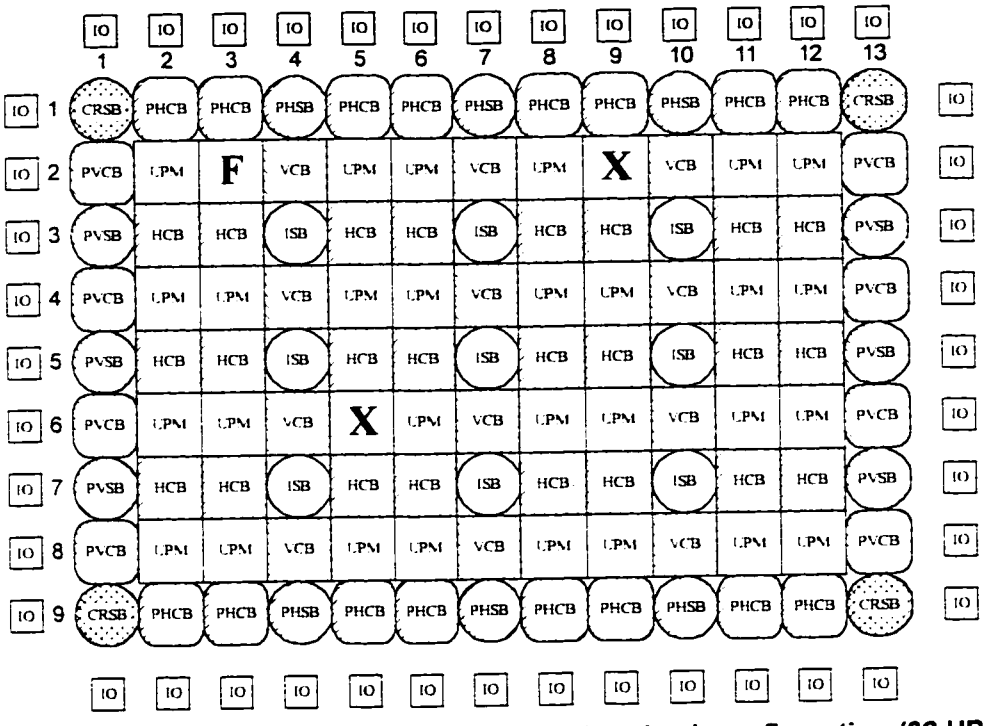


Figure 49: Three pins net on 9*13 FPGA with back-to-back configuration (32 UPMs).

First, the FPGA is mapped onto a 2-D array (Figure 50). Then, $UPM_{2,3}$ is labeled as initial source because it has an output pin on the net (driver pin). The other two UPMs are labeled as targets (Figure 50). The propagation starts from the source with zero. Since the output F is available on the top, right and bottom sides, the CBs on these

three sides are checked. Assuming the FPGA is lightly congested, the three CBs are free and labeled with 0+1. The propagation step then continues as described earlier. Figure 51 shows the first six iterations on propagation for the first time.

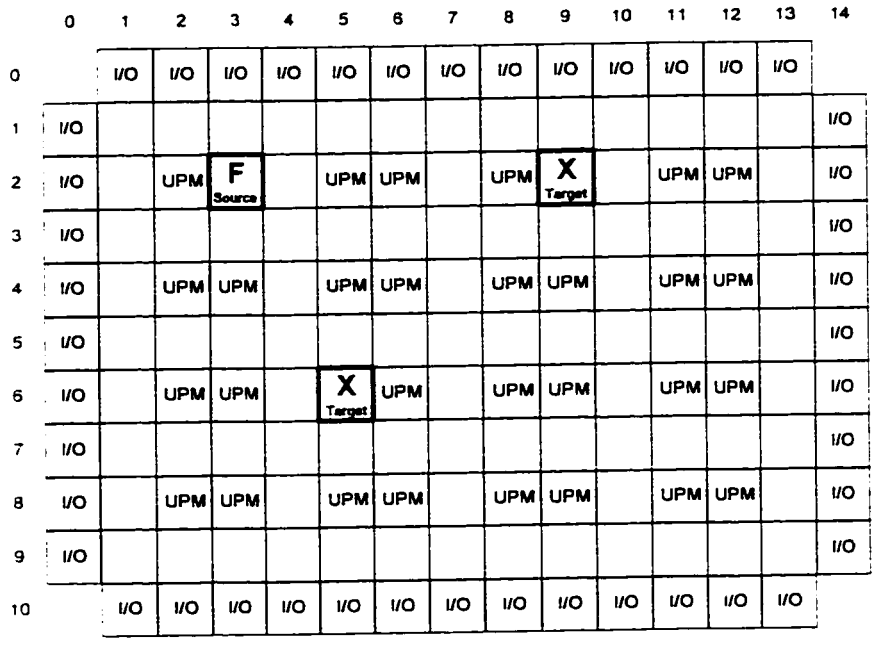


Figure 50: Mapping the FPGA into 2-D Array and specifying Source and Targets

In the sixth iteration, a UPM that contains a target was reached ($UPM_{6,5}$). The UPM on the left has five logically equivalent pins on the left, top and bottom sides. Since $UPM_{6,5}$ is on the left, and it is seen from its upper and left sides, a target was found and retracing will start. Four paths were found from $UPM_{6,5}$ to the source $UPM_{2,3}$ (Figure 52). The paths are:

1. $UPM_{6,5}$, $CB_{5,5}$, $SB_{5,4}$, $CB_{4,4}$, $SB_{3,4}$, $CB_{2,4}$, $UPM_{2,3}$
2. $UPM_{6,5}$, $CB_{5,5}$, $SB_{5,4}$, $CB_{4,4}$, $SB_{3,4}$, $CB_{3,3}$, $UPM_{2,3}$
3. $UPM_{6,5}$, $CB_{6,4}$, $SB_{5,4}$, $CB_{4,4}$, $SB_{3,4}$, $CB_{2,4}$, $UPM_{2,3}$
4. $UPM_{6,5}$, $CB_{6,4}$, $SB_{5,4}$, $CB_{4,4}$, $SB_{3,4}$, $CB_{3,3}$, $UPM_{2,3}$

The distance from the driving pin for all the four paths is equal to 6.

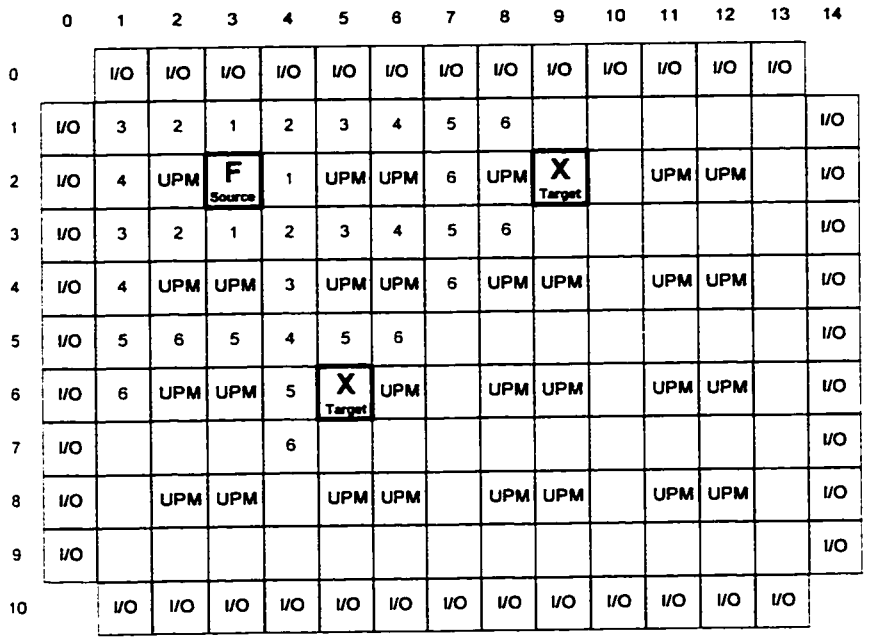


Figure 51: First six iterations on propagation for the first time.

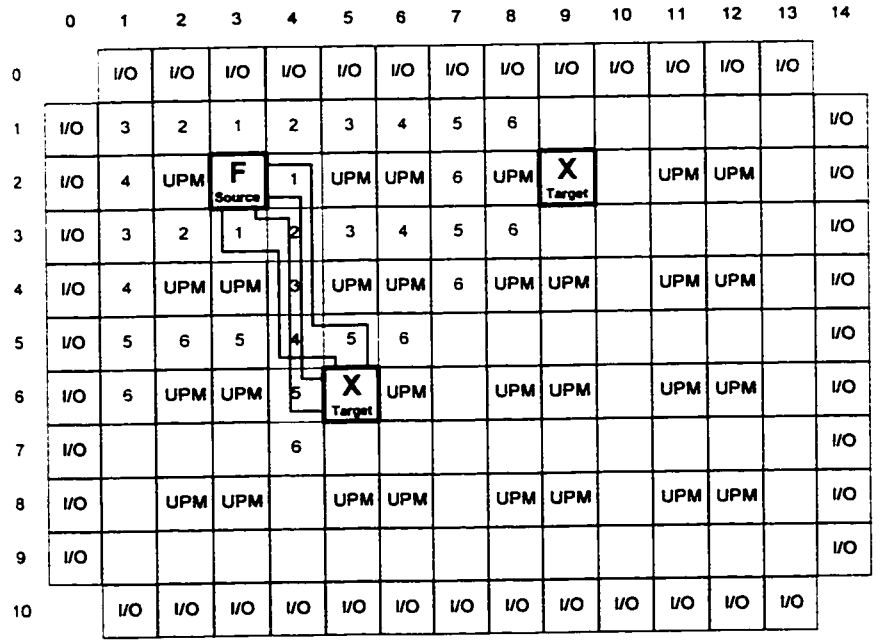


Figure 52: Retracing to find all paths from $UPM_{6,5}$ to a source.

Since not all net pins are found, the propagation continues (Figure 53). After two more iterations, $UPM_{2,9}$ is reached and the retrace procedure will start to find all paths from $UPM_{2,9}$ to the source. In this example, four paths were found from $UPM_{2,9}$ to the source $UPM_{2,3}$ (Figure 54). The paths are:

5. $UPM_{2,9}$, $CB_{1,9}$, $CB_{1,8}$, $SB_{1,7}$, $CB_{1,6}$, $CB_{1,5}$, $SB_{1,4}$, $CB_{1,3}$, $UPM_{2,3}$
6. $UPM_{2,9}$, $CB_{1,9}$, $CB_{1,8}$, $SB_{1,7}$, $CB_{1,6}$, $CB_{1,5}$, $SB_{1,4}$, $CB_{2,4}$, $UPM_{2,3}$
7. $UPM_{2,9}$, $CB_{3,9}$, $CB_{3,8}$, $SB_{3,7}$, $CB_{3,6}$, $CB_{3,5}$, $SB_{3,4}$, $CB_{3,3}$, $UPM_{2,3}$
8. $UPM_{2,9}$, $CB_{3,9}$, $CB_{3,8}$, $SB_{3,7}$, $CB_{3,6}$, $CB_{3,5}$, $SB_{3,4}$, $CB_{2,4}$, $UPM_{2,3}$

The distance from the driving pin for all the 4 paths equal to 8.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0		I/O	I/O	I/O	I/O	I/O	I/O	I/O	I/O	I/O	I/O	I/O	I/O	I/O	
1	I/O	3	2	1	2	3	4	5	6	7	8				I/O
2	I/O	4	UPM	F Source	1	UPM	UPM	6	UPM	X Target		UPM	UPM		I/O
3	I/O	3	2	1	2	3	4	5	6	7	8				I/O
4	I/O	4	UPM	UPM	3	UPM	UPM	6	UPM	UPM		UPM	UPM		I/O
5	I/O	5	6	5	4	5	6	7	8						I/O
6	I/O	6	UPM	UPM	5	X Target	UPM	8	UPM	UPM		UPM	UPM		I/O
7	I/O	7	8	7	6	7	8								I/O
8	I/O	8	UPM	UPM	7	UPM	UPM		UPM	UPM		UPM	UPM		I/O
9	I/O				8										I/O
10		I/O	I/O	I/O	I/O	I/O	I/O	I/O	I/O	I/O	I/O	I/O	I/O	I/O	

Figure 53: Continuing propagation to find the remaining nodes.

All the net pins are thus reached, the algorithm will find the cost of all paths. The first element in the cost function is congestion. Since the FPGA is assumed lightly congested, the congestion costs for all the eight paths are almost zero ($\mu_{Low\_con} = 1$).

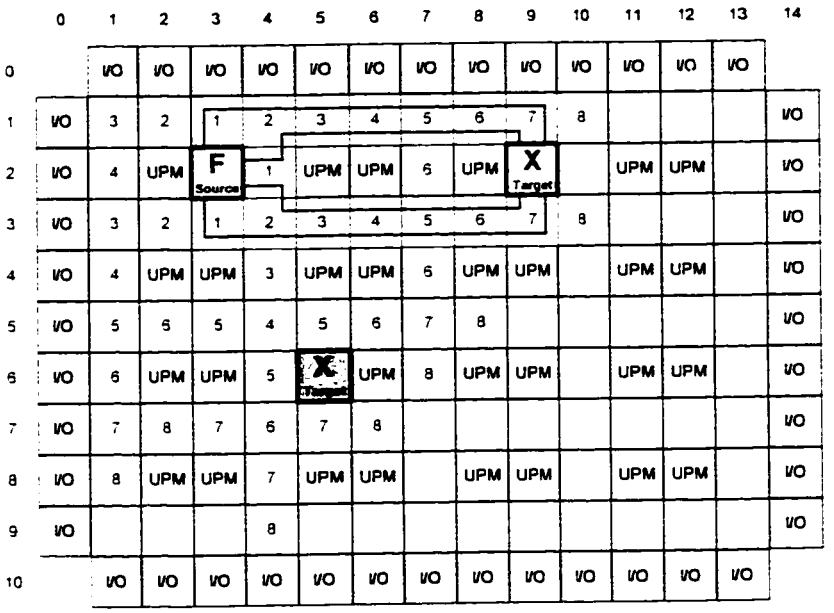


Figure 54: Retracing to find all paths from $UPM_{2,9}$ to a source.

The second element in the cost function is the average distance to the yet unconnected net pins. The distance is equal to the sum of Manhattan distance from all blocks in the path (source and target are not in the path) to the remaining unconnected net pins. The distance calculation for the 8 paths is given in Table 1. The short distance membership function for this example in shown in Figure 55.

The third element in the cost function is the path delay, which is a function of the number of blocks used by the path and number of bends on it. The delay calculation using the fuzzy function described in the previous section (Figure 48, page 68) for the eight paths is given in Table 2.

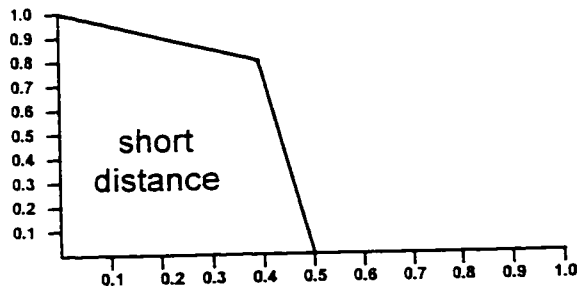
Table 1: Distance From Other Nodes to All Path Blocks.

(a)

Path	Distance from UPM$_{n,s}$ to block					sum	average by 5	norm by (9+13)*1	μ_{Low_dist}
1	2,4 5	3,4 6	4,4 7	5,4 8	5,5 7	33	6.6	0.30	0.85
2	3,3 7	3,4 6	4,4 7	5,4 8	5,5 7	35	7	0.32	0.84
3	2,4 5	3,4 6	4,4 7	5,4 8	6,4 9	35	7	0.32	0.84
4	3,3 7	3,4 6	4,4 7	5,4 8	6,4 9	37	7.4	0.34	0.83

(b)

Path	Distance from UPM$_{n,s}$ to block							sum	average by 7	norm by (9+13)*1	μ_{Low_dist}
5	1,3 7	1,4 6	1,5 5	1,6 6	1,7 7	1,8 8	1,9 9	48	6.85	0.31	0.85
6	2,4 5	1,4 6	1,5 5	1,6 6	1,7 7	1,8 8	1,9 9	46	6.57	0.30	0.85
7	3,3 5	3,4 4	3,5 3	3,6 4	3,7 5	3,8 6	3,9 7	34	4.85	0.22	0.89
8	2,4 5	3,4 4	3,5 3	3,6 4	3,7 5	3,8 6	3,9 7	34	4.85	0.22	0.89



$$\mu_{Short\_dist}(x) = \begin{cases} 1 - \frac{x}{2} & 0 \le x \le 0.4 \\ 4 - 8x & 0.4 \le x \le 0.5 \\ 0 & 0.5 \le x \le 1 \end{cases}$$

Where $x = Dist(path)$

Figure 55: *Short distance* membership function.

Table 2: Delay Cost of Various Paths.

Path number	# of blocks	# of binds	delay cost α=1, β=0.5	norm by 33 1.5*(9+13)	μ_{Low_delay}
1	5	1	5.5	0.17	0.83
2	5	2	6	0.18	0.82
3	5	0	5	0.15	0.85
4	5	1	5.5	0.17	0.83
5	7	0	7	0.21	0.72
6	7	1	7.5	0.23	0.65
7	7	0	7	0.21	0.72
8	7	1	7.5	0.23	0.65

The total cost for all paths using the additive combiner operator (Figure 44, page 64) with the following parameters is shown in Table 3.

$$\mu_{Low\_cost} = 0.3*\mu_{Low\_con} + 0.3*\mu_{short\_dist} + 0.4*\mu_{Low\_delay}$$

Table 3: Total Cost.

Path number	μ_{Low_con}	μ_{Low_dist}	μ_{Low_delay}	μ_{Low_cost}
1	1	0.85	0.83	0.887
2	1	0.84	0.82	0.880
3	**1**	**0.84**	**0.85**	**0.892**
4	1	0.83	0.83	0.881
5	1	0.85	0.72	0.843
6	1	0.85	0.65	0.815
7	1	0.89	0.72	0.855
8	1	0.89	0.65	0.855

The path with the minimum cost is number 3

3. $UPM_{6,5}$, $CB_{6,4}$, $SB_{5,4}$, $CB_{4,4}$, $SB_{3,4}$, $CB_{2,4}$, $UPM_{2,3}$

which is considered as the first branch in the tree. The resources should be updated. The following actions are taken:

1. $UPM_{6,5}$: number of available inputs on the left side is decremented by 1.
2. $CB_{6,4}$, $CB_{4,4}$, and $CB_{2,4}$: number of available tracks is decremented by 1.
3. $SB_{5,4}$, and $SB_{3,4}$: number of available connections from top to bottom is decremented by 1. In addition, the number of available pins on top and on bottom is decremented by 1.
4. $UPM_{2,3}$: number of available output on the right side is decremented by 1

Next, the algorithm checks if all net pins are connected. Since there is still one more unconnected pin, the propagation will start again. All labeled blocks are cleared. The sources are chosen from the solution tree blocks. All SBs and CBs on the solution tree are considered sources. The initial value of these blocks equal to the number of

blocks traversed to reach the block from the initial source $UPM_{2,3}$ (Figure 56). $UPM_{2,3}$ is considered as one of the sources because the net pin connected to it is an output pin. However, $UPM_{5,6}$ is not marked as a source since its pin in the net is input.

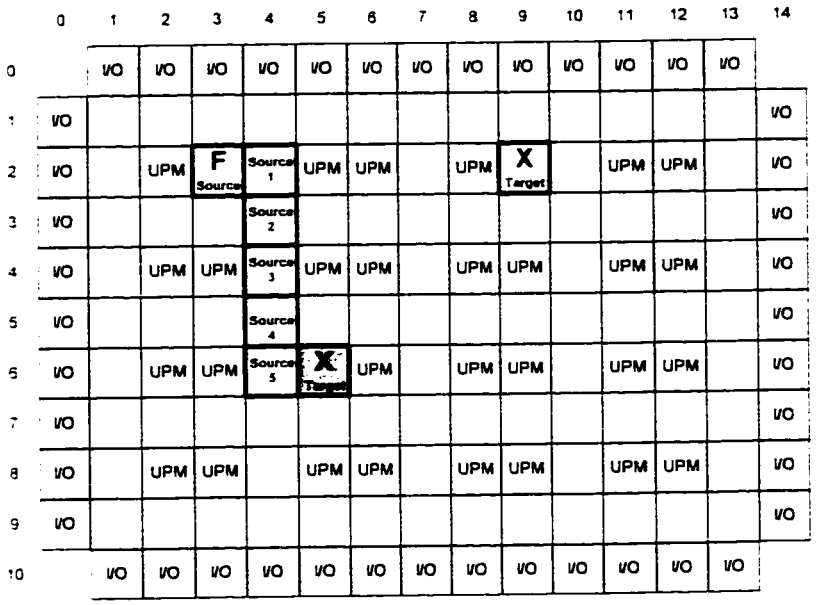


Figure 56: Identifying the solution tree blocks as source.

The propagation is then restarted to find the remaining net pins (Figure 57). Note that during propagation, the algorithm will propagate from all sources simultaneously by labeling the adjacent cells with the source value plus one. A target is found once $UPM_{2,9}$ is reached (Figure 58). Thus, the retracing process is triggered to find all possible paths from the target to one of the source blocks. Three paths were found from $UPM_{2,9}$ to the sources (Figure 59). The paths are:

1. $UPM_{2,9}$, $CB_{1,9}$, $CB_{1,8}$, $SB_{1,7}$, $CB_{1,6}$, $CB_{1,5}$, $SB_{1,4}$, $CB_{1,3}$, $UPM_{2,3}$
2. $UPM_{2,9}$, $CB_{1,9}$, $CB_{1,8}$, $SB_{1,7}$, $CB_{1,6}$, $CB_{1,5}$, $SB_{1,4}$, $CB_{2,4}$
3. $UPM_{2,9}$, $CB_{3,9}$, $CB_{3,8}$, $SB_{3,7}$, $CB_{3,6}$, $CB_{3,5}$, $SB_{3,4}$

The distance from the driving pin for all the three paths equal to 8.

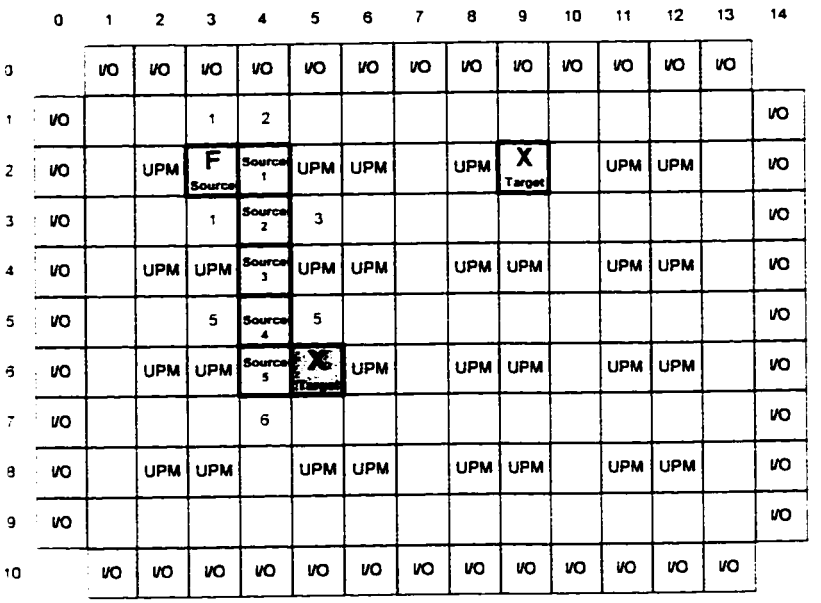


Figure 57: First iteration in propagation for the second time.

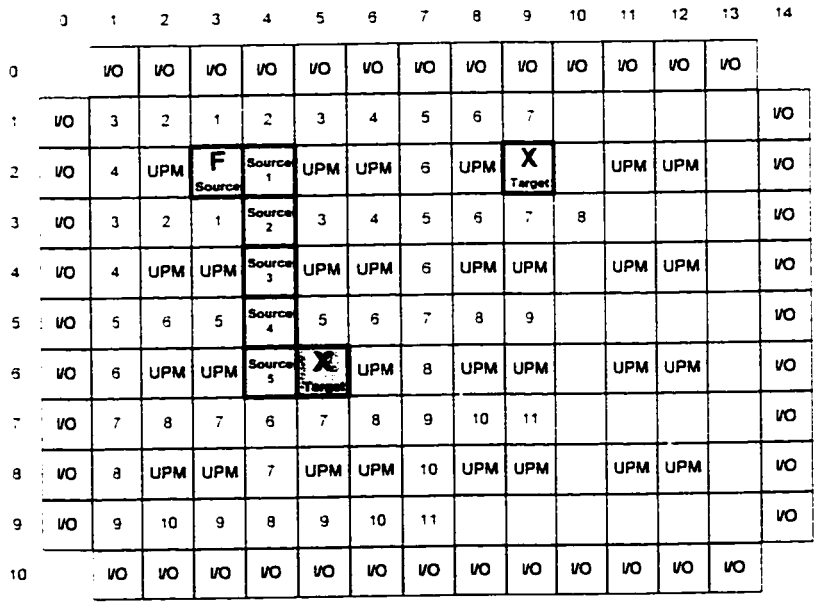


Figure 58: Propagation for the second time.

All net pins are thus reached and the algorithm finds the cost of each path. The first element in the cost function is congestion. The FPGA is still lightly congested therefore, the congestion cost for all 3 paths is almost zero (CON=0 , $\mu_{Low\_con} = 1$).

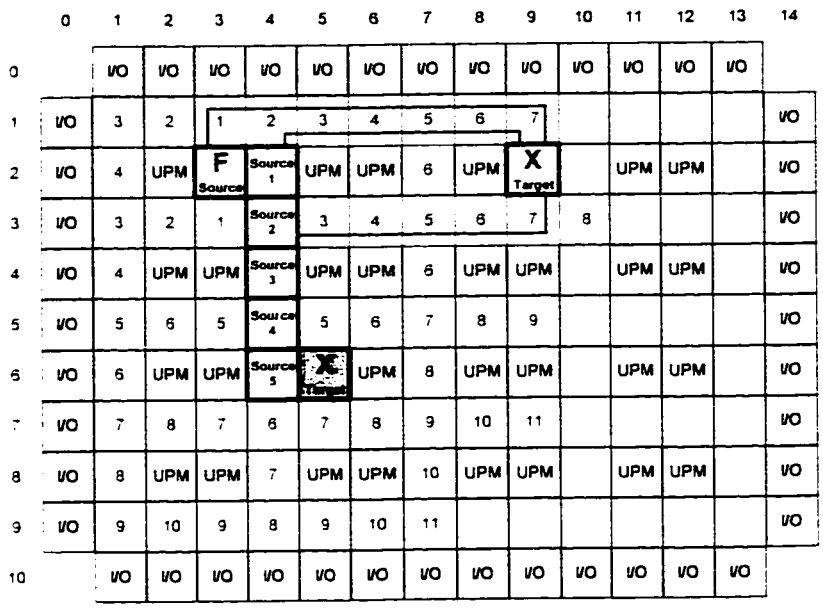


Figure 59: Retracing for the second time.

The second element is the average distance to the yet unconnected net pins. Since all net pins are connected (have been reached), the distance is equal to zero for all paths (Dist=0, μ_{short_dist}=1). The third element in the cost function, the delay sum of the three paths, is given in Table 4. The total cost for all paths using the additive combiner operator (Figure 44, page 64) is shown in Table 5.

Table 4: Delay Cost of Various Paths.

Path number	# of blocks	# of binds	delay cost α=1, β=0.5	norm by 33 1.5*(9+13)	μ_{low_delay}
1	7	0	7	0.21	0.72
2	6	1	6.5	0.20	0.80
3	5	1	5.5	0.17	0.83

Table 5: Total Cost.

Path number	μ_{low_con}	μ_{low_dist}	μ_{low_delay}	μ_{low_cost}
1	1	1	0.72	0.888
2	1	1	0.80	0.920
3	**1**	**1**	**0.83**	**0.932**

The path with the minimum cost is number 3.

3. $UPM_{2,9}$, $CB_{3,9}$, $CB_{3,8}$, $SB_{3,7}$, $CB_{3,6}$, $CB_{3,5}$, $SB_{3,4}$

This path is considered as the second branch in the tree. The resources should be updated. The following actions are taken:

1. $UPM_{2,9}$: number of available inputs on bottom side is decremented by 1.
2. $CB_{3,9}$, $CB_{3,8}$, $CB_{3,6}$ and $CB_{3,5}$: available tracks is decremented by 1.
3. $SB_{3,4}$: number of available connections from left to right side is decremented by 1. In addition, the number of available pins on left and right sides are decremented by 1.
4. $SB_{3,7}$: number of available pins on right side is decremented by 1.

Since all net pins have been connected, the solution tree is complete. The algorithm then reports the solution tree for the current net. The Steiner tree that connects all three pins of the net is shown in Figure 60.

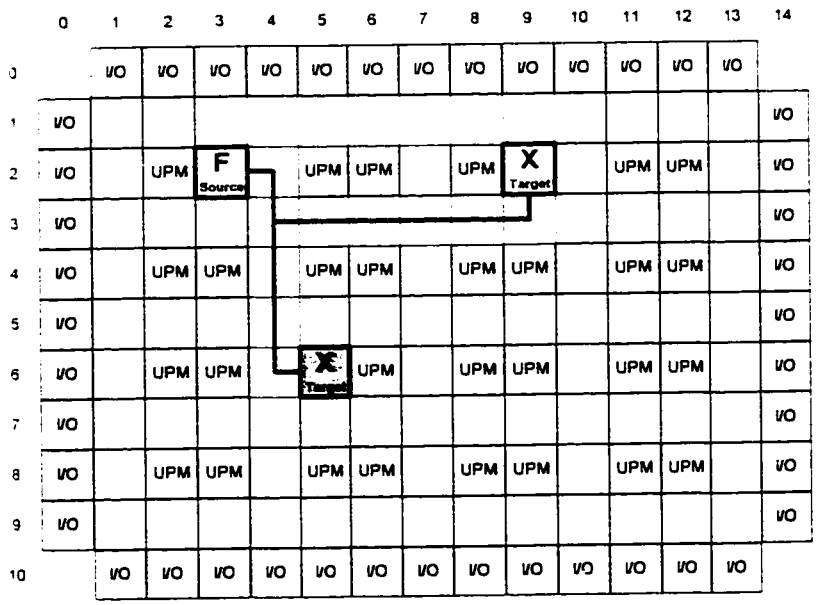


Figure 60: Steiner tree found by the global router to connect the three pins of the net.

Next, the algorithm proceeds and selects another net to route. After finishing all nets, the global router would have provided all information needed by the detailed router for all nets. The output of the global router for the above example is as follow:

- The maximum distance between the driving pin and one of the sinks is 8.
- The various branches of the solution tree are:
  1st segment: $UPM_{2,3}$, $CB_{2,4}$, $SB_{3,4}$, $CB_{4,4}$, $SB_{5,4}$, $CB_{6,4}$, $UPM_{6,5}$.
  2nd segment: $SB_{3,4}$, $CB_{3,5}$, $CB_{3,6}$, $SB_{3,7}$, $CB_{3,8}$, $CB_{3,9}$, $UPM_{2,9}$.

3.4 Complexity Analysis

In this section, the space and time requirements of the global router are analyzed. Throughout the analysis, we shall be using the following notations.

- R and C are the numbers of FPGA rows and columns respectively. The maximum array size (R* C) for the adopted architecture is 61*61.
- N is the number of nets. Maximum N for the adopted architecture is 1000.
- M is the maximum number of pins per net (the net size). Typically, M ranges from 5 to 10.
- ML is the maximum length of any path between two nodes[1]. Upper bound for ML is 2(R+C).
- A is the maximum number of alternative paths between two nodes. Practical estimate for A ranges between 5 to 10.
- Memory location and storage element are used interchangeably and both refer to 1 byte (8 bits).

[1] Node is either a net pin or a Steiner node.

3.4.1 Space Complexity

The global router needs to store information regarding the FPGA platform used as well as store the design netlist information. The storage requirements for the FPGA platform is independent of the array size (R*C). Storage requirements for the design netlist is, however, related to the FPGA array size. The algorithm extracts information from the FPGA template and stores them in special data structure (Figure 61).

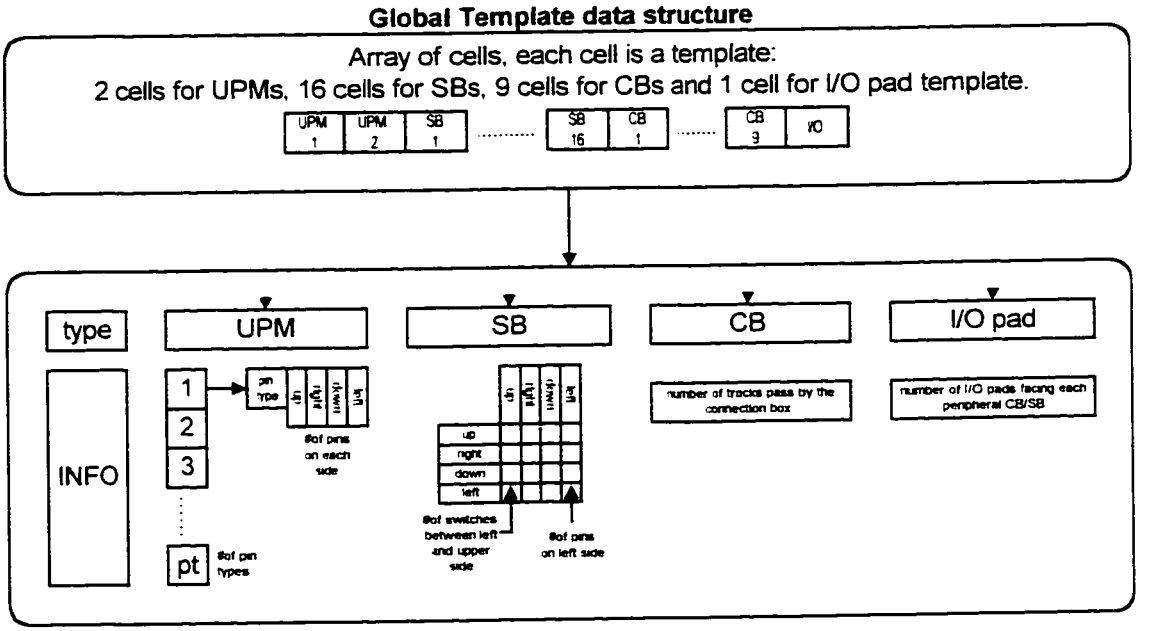


Figure 61: Global template data structure.

The storage requirements for the FPGA template (Figure 61) is as follows:

1. Each UPM needs five memory locations for each pin type. Therefore, 25 locations are required for the five different pin-types. For the back-to-back FPGA configuration, two UPM templates are needed (50 locations).
2. Each CB needs only one memory location. Therefore, six memory locations are needed to store the required data for CBs in the single UPM

configuration while nine locations are required for the back-to-back configuration (section 2.2.2.3, page 37) .

3. Each SB needs 16 memory locations to store the required information. A total of 256 (16*16) locations are required for the 16 different SB templates (section 2.2.2.4, page 40).
4. The I/O needs only one location.

Therefore, the global router needs to store information for 24 templates for single UPM FPGA configuration (Figure 32, page 47) or 28 templates for back-to-back FPGA (Figure 33, page 48). This requires 25+256+6+1 = 288 memory locations for single UPM configuration or 50+256+9+1 = 316 memory locations for back-to-back.

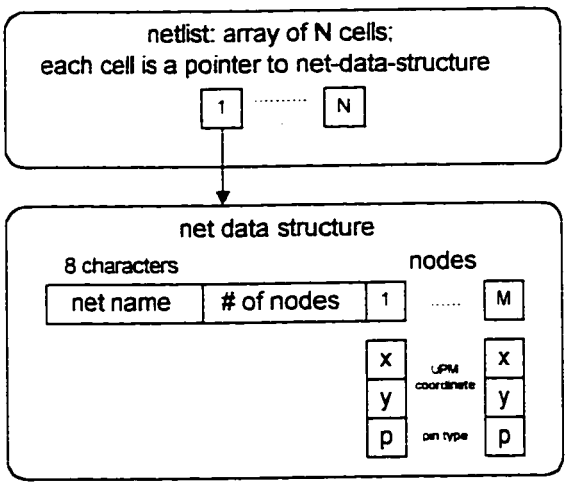


Figure 62: Netlist data structure.

In addition to the FPGA templates, the global router needs to store the design netlist. The netlist data structure is shown in Figure 62. As can be seen from the figure, each net requires eight locations for the net name, one location for the number of nodes in the net and three locations for each node. This adds up to 9+3M for each net. Thus, N nets require N*(9+3*M). Since M is relatively small, netlist requires O(N) locations.

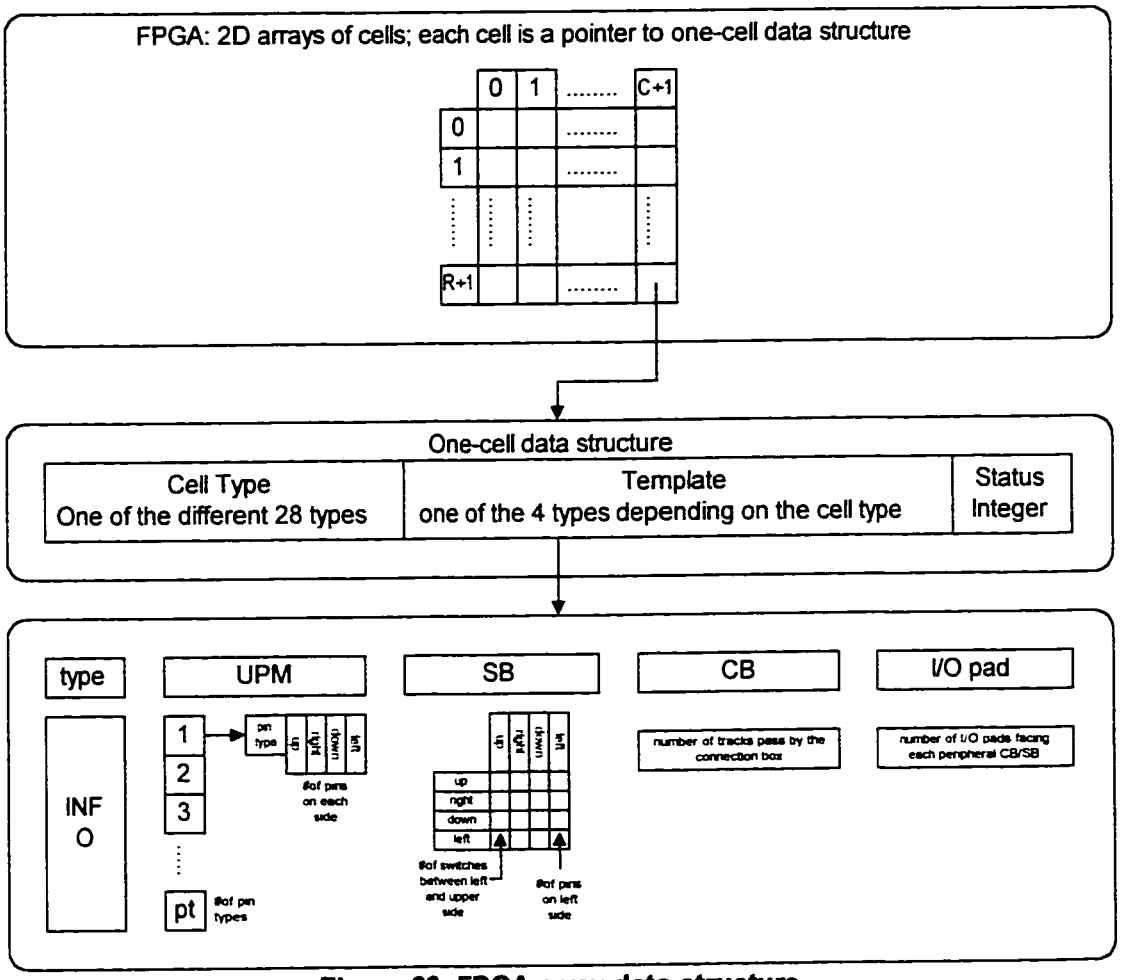


Figure 63: FPGA array data structure.

The FPGA is modeled as a 2-D array of size equal to (R+2)*(C+2) (Figure 63). Each cell in the array points to some structure containing information about the cell: cell type, available resources, and status. The cell type field is used to classify the cell into one of the available types (24 types for single UPM configuration and 28 for back-to-back). The available resources field for a cell is initially a copy of the global template of that cell. As the algorithm proceeds and resources are used, the available resources field is adjusted accordingly. The status field is an integer number indicating whether

the cell is a source, target, obstacle or free cell. The status field is used to store the cell label as well. The cell type and status require two memory locations each (total of 2(R+1)(C+1)= 2RC+2R+2C+2). Cell template space depends on the cell type. The UPM requires 25 locations, the SB requires 16, and the CB and the I/O require one each. The single UPM FPGA with R rows and C columns have:

- $\left(\frac{R-1}{2}\right)\left(\frac{C-1}{2}\right)=\frac{RC-R-C+1}{4}$ UPMs,
- $\left(\frac{R+1}{2}\right)\left(\frac{C+1}{2}\right)=\frac{RC+R+C+1}{4}$ SBs,
- $\left(\frac{R-1}{2}\right)\left(\frac{C+1}{2}\right)=\frac{RC+R-C-1}{4}$ Vertical CBs,
- $\left(\frac{R+1}{2}\right)\left(\frac{C-1}{2}\right)=\frac{RC-R+C-1}{4}$ Horizontal CBs, and
- $2(R+C)$ I/O pads

The space requirements for all templates is

$$25\frac{RC-R-C+1}{4}+16\frac{RC+R+C+1}{4}+\frac{RC+R-C-1}{4}+\frac{RC-R+C-1}{4}+2R+2C=$$

$$\frac{25RC+16RC+2RC-25R+16R+8R-25C+16C+8R+25+16-2}{4}=$$

$$\frac{43RC-R-C+39}{4}$$

Therefore, the FPGA model requires $2RC+2R+2C+2+\frac{43RC-R-C+39}{4}=$

$$\frac{8RC+8R+8C+8}{4}+\frac{43RC-R-C+39}{4}=\frac{51RC+7R+7C+47}{4}$$ memory locations.

Similarly, the back-to-back FPGA with R rows and C columns have:

- $\left(\frac{R-1}{2}\right)\left(2\frac{C-1}{3}\right)=\frac{2RC-2R-2C+2}{6}$ UPMs,
- $\left(\frac{R+1}{2}\right)\left(\frac{C+2}{3}\right)=\frac{RC+2R+C+2}{6}$ SBs,
- $\left(\frac{R-1}{2}\right)\left(\frac{C+2}{3}\right)=\frac{RC+2R-C-2}{6}$ Vertical CBs,
- $\left(\frac{R+1}{2}\right)\left(2\frac{C-1}{3}\right)=\frac{2RC-2R+2C-2}{6}$ Horizontal CBs, and
- $2(R+C)$ I/O pads

The space requirement for all templates is

$$25*\frac{2RC-2R-2C+2}{6}+16*\frac{RC+2R+C+2}{6}+\frac{3RC+C-4}{6}+2(R+C)=$$

$$\frac{50RC+16RC+3RC-50R+32R+12R-50C+16C+C+12C+50+32-4}{6}=$$

$$\frac{69RC-6R-21C+78}{6}$$

Therefore, the FPGA model requires $2RC+2R+2C+2+\frac{69RC-6R-21C+78}{6}=$

$$\frac{12RC+12R+12C+12}{6}+\frac{69RC-6R-21C+78}{6}=$$

$\frac{81RC+6R-9C+90}{6}$ memory locations.

During the propagation step, the algorithm needs to store the currently filled cells and the newly filled ones. The maximum number of cells to be filled in one iteration is equal to 2(R+C). While retracing, the algorithm stores all found paths. Each path requires a maximum of ML locations to store the cells and three memory locations to

store the cost function elements. The algorithm finds and stores paths to all net pins and selects the lowest cost path. Therefore, storing paths will need a maximum of (ML+3)*A*M locations.

Only the solution tree of the current net needs to be stored. The tree needs ML locations to store each branch and it has a maximum of M branches. Therefore, the required locations are M*ML.

From all of the above, the space requirements of the global router is of O(N+RC+R+C) or simply O(N+RC).

3.4.2 Time Complexity

The global router performs several steps for each net. In the worst case, propagation traverses all FPGA cells. Therefore, propagation timing is O(R*C). The retracing step might need to traverse all cells as well.

The cost calculation has three parameters. The congestion and delay computations require checking a maximum of ML cells for each path. The estimation of distance requires M*ML calculations. The maximum number of paths to evaluate is at most A*M.

Therefore, for one iteration to find one branch in the net, the program will perform O(R*C+M(A+ML)) operations[1]. The algorithm performs a maximum of M iterations to find the solution tree for one net. Therefore, for N nets, the algorithm requires O(N*M(R*C+M(A+ML))) operations.

Since ML is practically equal to R+C while A and M have typical maximum value of 10, the time complexity is O(N*(R*C+R+C)) or simply O(N*R*C).

[1] Operation refers to either a logical or arithmetical operation

CHAPTER 4

DETAILED ROUTER

The detailed routing step follows global routing. The task of the detailed router is to identify the exact tracks and switches needed to physically connect all nets in a given design. The detailed router should minimize the delay of the physical path.

To perform this task, the detailed router should be supplied with the FPGA general parameters (section 2.2.1, page 34), namely; the routing resources detailed structure and a coarse Steiner tree for each net. The routing resources details required by the detailed router are:

- For UPMs: the pin type of all pins on each side.
- For CBs: for each track, all pins that can be connected to it.
- For SBs: for each pin, all the possible connections from this pin to any other pin. This connection might be via a wire segment or a switch. Bypass tracks are added to the corresponding pin as a wire segment, however, the programmable switches on the switch box are added as switches.
- For I/O pads: types of all pins

The detailed router provides the required state (ON/OFF) of each switch in the routing resources (CBs and SBs) which implements the required design.

The used algorithm is based on the Coarse Graph Expansion (CGE) algorithm [16]. An overview of CGE algorithm is given next. Then, the description of the detailed routing algorithm is described. Then, the algorithm is illustrated by an example. Finally, the algorithm time and space complexities are analyzed.

4.1 CGE Algorithm

The global router assigns nets to a sequence of modules (UPMs, CBs, and SBs). This assignment is a coarse one. A module in such a sequence usually has more than one choice to connect a signal from a predecessor module to a successor module.

Consider the coarse graph in Figure 64-a as the output of the global router to connect $UPM_{2,3}$ with $UPM_{4,5}$. In Figure 64-b, the coarse graph is expanded according to the possibilities available to propagate the signal from one module to another. As it can be seen from the figure, $CB_{2,4}$ can connect the signal from the $UPM_{2,3}$ to $SB_{3,4}$ with three different possibilities. There are five different paths to connect the two UPMs.

CGE is used to find all possibilities available to connect a 2-node sub-net and choose

the one with minimum cost. The algorithm has two phases: (1) expansion of the coarse graph, and (2) connection formation [16].

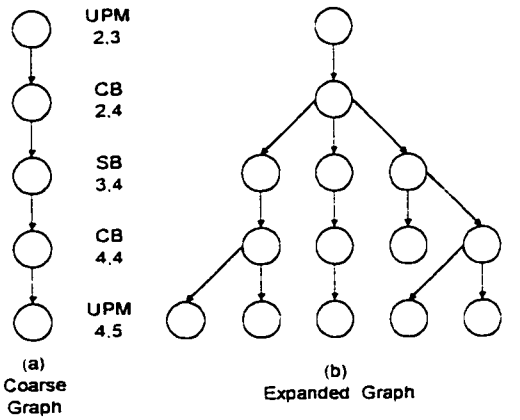


Figure 64: A coarse graph and its expansion.

During the expansion phase, the algorithm expands all the coarse graphs of a given design and finds all possible paths to connect the two nodes each coarse graph represents. The paths are stored in a list and passed to the following connection formation phase [16].

In the second phase, the connection formation, the algorithm takes the list constructed during phase one and finds the cost of all paths in the list. A choice of a certain path may affect the cost and validity of some paths in the list. A path on the list is no longer valid if it conflicts with the selected path. Path cost is affected because the cost depends on other paths in the list. If path p is selected, the algorithm removes all paths conflicting with p from the list [16].

4.2 Detailed Router Algorithm

The detailed router task is to specify the actual wire segments and switches to be used for connecting each net in the design. The algorithm has two phases: expansion phase and connection formation phase. Next, both phases are described.

4.2.1 Expansion

This phase expands the coarse Steiner tree of each net and stores all possible expansions on *path-list*. This expansion algorithm is outlined in Figure 65.

```
Procedure Expansion
Begin
   For each net do
      For each branch in the solution tree do
         For all resources R in the first block B in the branch do
            if R can be part of the net then
               Expand(B,R)
            End-if
         End-For
      End-For
   End-For
End
```

Figure 65: Expansion procedure.

Resource R can be part of the net if it represents:

1. a pin on a UPM or an I/O pad in the net of the desired type.
2. a track on a CB that is already used by one of the branches of the current net solution tree, or
3. a switch box pin that can be connected to an already used track by one of the current net solution tree branches.

The procedure *Expand* is shown in Figure 66. The procedure could have some expansion limit to save time and space. After the expansion phase is completed, the *path-list* is passed to the next phase.

```
Procedure Expand(B1,R1)
Begin
   If B1 is the last block in the tree then
      If R1 can be part of the net then
         Push (R1,B1) to Stack
         Store Stack in the path-list
      End-If
   Else
      for all resources R2 in the successor of B1 (B2) in the branch do
         If R2 can be part of the net then
            Push (R1,B1) to Stack
            Expand (B2,R2)
         End-If
      End-For
   End-If
END
```

Figure 66: Expand procedure.

```
Procedure Connection Formation
Begin
   Find cost of all paths in the path-list
   While (path-list is not empty) and (not all nets are connected) do
      If there is essential paths in path-list then
         Choose the essential path EP with minimum cost
         Remove all paths conflicting with EP from path-list
      Else if there is critical paths in path-list then
         Choose the critical path CP with minimum cost
         Remove all paths conflicting with CP from path-list
      Else
         Choose the path P with minimum cost
         Remove all paths conflicting with P from path-list
      End-If
      Update paths cost
   End-While
End
```

Figure 67: Connection formation procedure.

4.2.2 Connection Formation

During this phase, a path from the *path-list* is selected according to the path priority

and cost. Essential paths, paths that have no alternative in the *path-list*, are given the highest priority. The critical path, path that corresponds to a branch on the solution tree of a critical net, is given priority lower than essential paths but higher than non-critical paths. The path cost calculation is described in the following subsection. The connection formation procedure is outlined in Figure 67.

4.2.3 Cost Function

The cost function is chosen to guide the algorithm in selecting paths that have relatively small negative effect on the remaining connections [16]. Furthermore, the selected paths should have acceptably small delays.

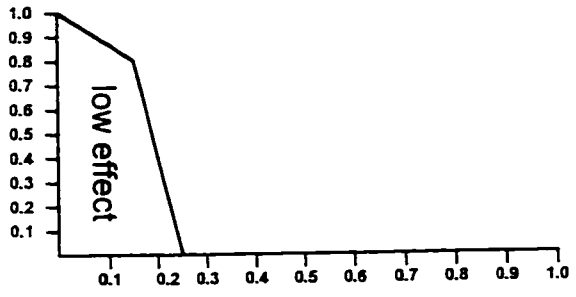
4.2.3.1 Effect on Remaining Paths

If a path is assigned a particular route, the unused resources available to route the remaining paths are reduced. It is best if the chosen route has a minimal effect on the routability of the remaining paths. Such effect is estimated by the following function:

$$\mathrm{Eff}(p) = \underset{\forall e \in p}{\mathrm{MAX}}\, \mathrm{alt}(e)$$

where alt(e) is the number of paths in the *path-list* that share segment or switch e with path p. The *Eff* function is normalized in the interval [0,1] by the number of paths in the *path-list*. The path should not conflict with more than a pre-specified percentage of the remaining paths in the *path-list*. Figure 68 illustrates the membership function

μ_{Low_Eff} of paths in the fuzzy subset of *low-effect* paths. The figure assumes that the path should not conflict with more than 25% of the remaining paths in the *path-list*



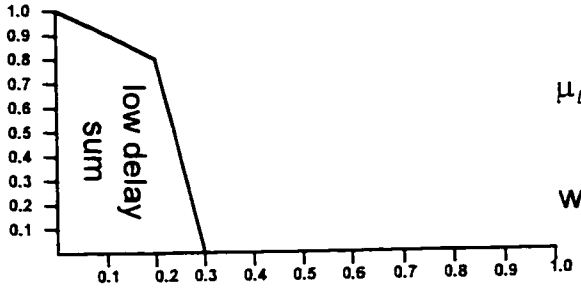
$$\mu_{Low\_Eff}(x) = \begin{cases} 1 - \frac{4}{3}x & 0 \le x \le 0.15 \\ 2 - 8x & 0.15 \le x \le 0.25 \\ 0 & 0.25 \le x \le 1 \end{cases}$$

where $x = Eff(path)$

Figure 68: Membership function for effect on other paths.

4.2.3.2 Delay Sum

The path delay is estimated as the sum of its individual component delays, which is calculated as $\text{Sum}(p) = \sum_{\forall e \in p} \text{delay}(e)$ where each element e on path p is either a wire segment or a switch. Figure 69 shows the membership function μ_{Low_Sum} of the fuzzy subset of *low-Sum* paths. The Figure assumes that the delay sum should not exceed 30% of the normalizing factor $(\alpha+\gamma)(R+C)$. α and γ are wire delay per unit length and SB switch delay respectively.



$$\mu_{Low\_Sum}(x) = \begin{cases} 1 - x & 0 \le x \le 0.2 \\ \frac{12}{5} - 8x & 0.2 \le x \le 0.3 \\ 0 & 0.3 \le x \le 1 \end{cases}$$

where $x = sum(path)$

Figure 69: Membership function for the delay sum.

4.2.3.3 Total Cost Membership Functions

The fuzzy cost of a path p is set equal to the value of the membership μ_{Low_Cost} of p in the fuzzy subset of low-cost paths with respect to the above two criteria:

$$\mu_{Low\_Cost} = w_1 * \mu_{Low\_Eff} + w_2 * \mu_{Low\_Sum}$$

where w1+w2 = 1. The best path is the path with the highest μ_{Low_Cost}.

4.3 Example

In this section, the detailed router is illustrated by an example. Consider the tree derived for the net in the global routing example (section 3.3). The tree is supplied to the detailed router as follows:

1st segment				2nd segment			
	2	3	2		3	4	
	2	4			3	5	
	3	4			3	6	
	4	4			3	7	
	5	4			3	8	
	6	4			3	9	
	6	5	1		2	9	1

The algorithm determines the block type from its coordinates (the first two numbers on each row). UPMs and I/O pads need, in addition to the coordinates, the pin-type (the third number on the row).

In the following text, the term branch and segment are used interchangeably. The algorithm starts expanding the coarse Steiner tree one segment at time. It expands the

first segment in the tree starting from $UPM_{2,3}$. $UPM_{2,3}$ is connected to vertical $CB_{2,4}$ on its right. Therefore, the algorithm expands all pins on the right side of $UPM_{2,3}$ of type 2 (output F). F is available on pin number 1 on the right side of the right UPM on the FPGA (Figure 70). The algorithm stores the triplicate (2,3,1) in a stack and proceeds to expand the next block ($CB_{2,4}$) from all resources connected to pin 1 on the right side of $UPM_{2,3}$.

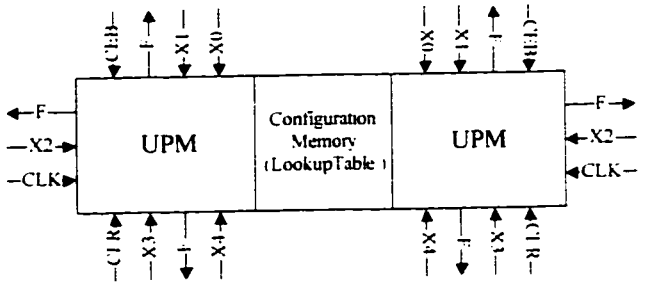


Figure 70: Two back-to-back UPMs.

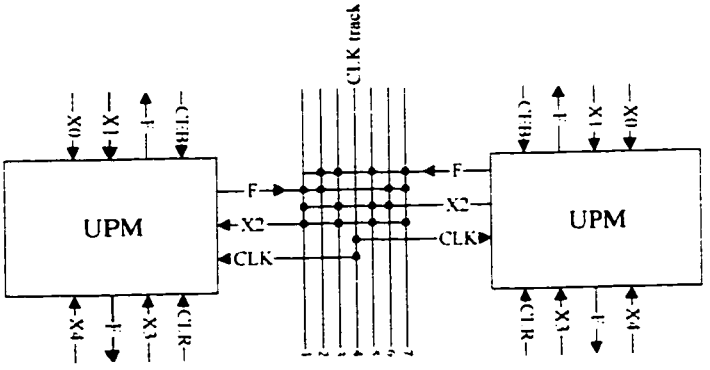


Figure 71: Internal vertical connection box.

As can be seen in Figure 71, pin 1 on the right side of the right UPM can connect to tracks 1,2,6 and 7 on the vertical channel. The algorithm will expand the path through all the possible tracks one by one. First, it stores the triplicate (2,4,1) and expands the

next block $SB_{3,4}$ from the pin connected to track 1 on the top SB side.

The path uses $SB_{3,4}$ to connect from top to bottom. Pin 1 on the upper side of the SB is traced for all switches connecting from this pin to any pin on the bottom side. As shown in Figure 72, pin 1 can be connected to pin 19 on the bottom side, which is a segment on track 5 on the vertical channel. The triplicate (3,4,19) is stored and the next block is checked. The process continues using the template of the three blocks shown in Figure 71, Figure 72 and Figure 73. The expansion is shown in Table 6.

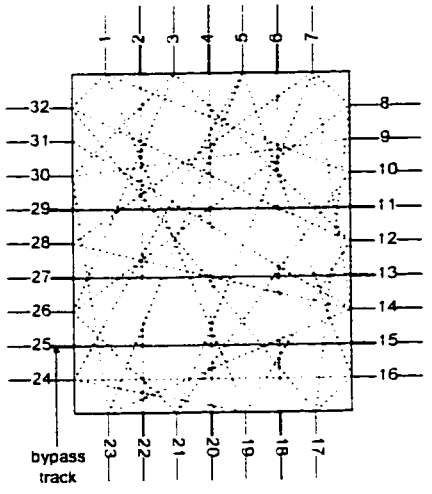


Figure 72: An internal even-even switch box and its template description.

Table 6: Expansion for Branch 1.

R-OPM 2,3	V-CB 2,4	EE-SB 3,4	V-CB 4,4	EO-SB 5,4	V-CB 6,4	L-OPM 6,5
1	1	19	5	19	5	2
1	3	22	2			
1	6	18	6			
1	7	21	3	21	3	2

As it can be seen from Table 6, the algorithm found two alternatives for the first segment. While expansion from $CB_{4,4}$ track 2, $SB_{5,4}$ does not have a switch (or a

bypass track) from top to bottom, therefore, the algorithm backtracks.

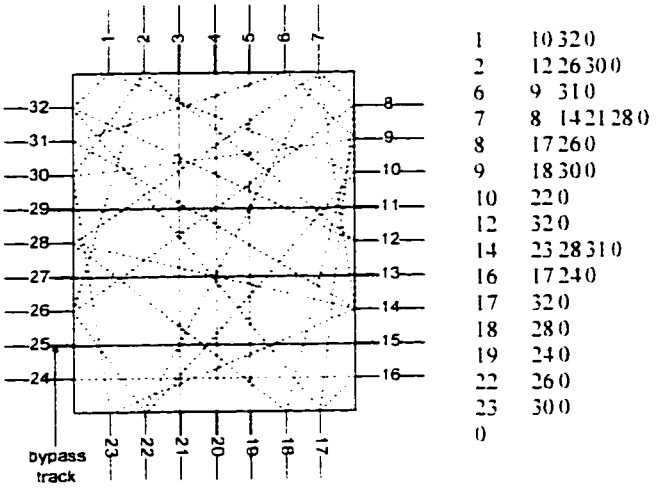


Figure 73: An internal even-odd switch box and its template description.

In a similar manner, the second branch is expanded using the information on Figure 70, Figure 72, Figure 74 and Figure 75. $SB_{3,4}$ is a Steiner node in the solution tree provided by the global router. The expansion starts from this block, from the pin used by the other solution tree branches. Branch 1 has 2 alternatives in which it uses pin 1 and 19 in one and 7 and 21 in the other. All alternative paths found starting from pin 1 or 19 are associated with the first alternative for branch 1. If one of such alternatives is to be chosen to make the connection, the first alternative for branch 1 should be chosen too. Similarly, the alternative paths start from either 7 or 21 are associated with the second alternative for branch 1.

The expansion is shown in Table 7. It can be seen that 8 alternatives were found: 3 associated with the first alternative of branch 1 and 5 with the second alternative. The

paths are stored in the path-list. The paths for the net "A" are shown in Table 8.

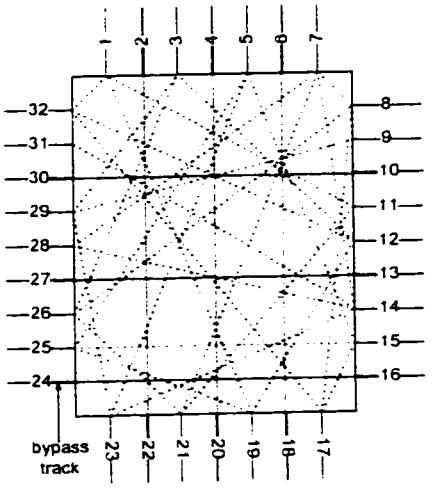


Figure 74: An internal odd-even switch box (OE-SB) and its template description.

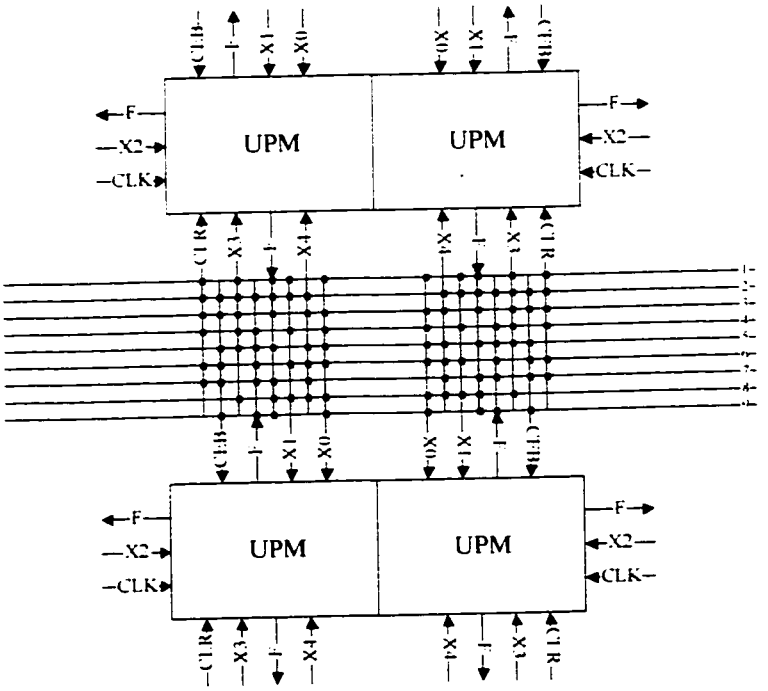


Figure 75: Two back-to-back internal horizontal connection boxes.

Table 7: Expansion for Branch 2.

EE-SB 3,4	LH-CB 3,5	RH-CB 3,6	OE-SB 3,7	LH-CB 3,8	RH-SB 3,9	R-UPM 2,9
1	(10)=3	3	10	3	3	2
						4
19	(12)=5	5	14	-	7	4
7	(8)=1	1	12	5	5	2
						4
	(14)=7	7	8	1	1	2
21	(10)=3	3	10	3	3	2
						4

Table 8: Path List.

Net name	Branch #	Alt. Num								Association
A	1	1	2,3,1	2,4,1	3,4,19	4,4,5	5,4,19	6,4,5	6,5,2	
		2	2,3,1	2,4,1	3,4,19	4,4,5	5,4,19	6,4,5	6,5,2	
	2	1	3,4,1	3,5,3	3,6,3	3,7,10	3,8,3	3,9,3	2,9,2	1,1
		2	3,4,1	3,5,3	3,6,3	3,7,10	3,8,3	3,9,3	2,9,4	1,1
		3	3,4,19	3,5,5	3,6,5	3,7,14	3,8,7	3,9,7	2,9,4	1,1
		4	3,4,7	3,5,1	3,6,1	3,7,12	3,8,5	3,9,5	2,9,2	1,2
		5	3,4,7	3,5,1	3,6,1	3,7,12	3,8,5	3,9,5	2,9,4	1,2
		6	3,4,7	3,5,7	3,6,7	3,7,8	3,8,1	3,9,1	2,9,2	1,2
		7	3,4,21	3,5,3	3,6,3	3,7,10	3,8,3	3,9,3	2,9,2	1,2
		8	3,4,21	3,5,3	3,6,3	3,7,10	3,8,3	3,9,3	2,9,4	1,2

Once the algorithm finishes expanding all the coarse Steiner trees, it proceeds to the connection formation phase. The costs of all paths are calculated first. The cost function has two components: the effect on other paths on the list and the delay sum. Assuming that the effect of all the paths for this example is very small then μ_{Low_Eff}=1. Table 9 shows the delay sum calculation of the paths on Table 8. The delay sum is equal to α*WS + γ*SBS + λ*CBS, where

α, γ, λ delay of unit wire segment, SB switch and CB switch respectively.

WS, SBS, CBS number of wire segment, SB switches and CB switches on the path (respectively).

After cost calculation, the algorithm selects paths (form connections) depending on the path cost. Some paths are given higher priority and they are selected first.

Table 9: Delay Sum.

Net name	Branch #	Alternativ e #	WS	SBS	CBS	Delay Sum α=1, γ=2, λ=1	Normalis ed Delay sum by 66	μ_{new_sum}
A	1	1	4	1	2	8	0.12	0.88
		2	4	1	2	8	0.12	0.88
	2	1	5	1	1	8	0.12	0.88
		2	5	1	1	8	0.12	0.88
		3	4	2	1	9	0.14	0.86
		4	4	2	1	9	0.14	0.86
		5	4	2	1	9	0.14	0.86
		6	4	2	1	9	0.14	0.86
		7	5	1	1	8	0.12	0.88
		8	5	1	1	8	0.12	0.88

4.4 Complexity Analysis

In this section, the space and time complexity of the detailed router are analyzed. First, some terms need to be defined

- R and C are the numbers of FPGA rows and columns respectively. Typical maximum array size (R* C) for the adopted architecture is 61*61.
- N is the number of nets. Maximum N for the adopted architecture is 1000.
- M is the maximum number of pins per net (net size). Typically, M ranges from 5 to 10.

- ML is the maximum length of any path between two nodes. Upper bound value for ML is 2(R+C).
- EL is the expansion limit, which is the maximum number of alternative paths between two nodes. Typical value for EL is 10 to 20.
- PPS is the maximum number of pins per UPM side (typically 4).
- PT is the number of different UPM pin-types (typically 5)
- P is the total number of SB pins. It depends on the size of routing channels.
- F is the maximum SB flexibility. Typically between 3 to 5.
- MT is maximum number of tracks per channel.
- IOP is the number of I/O pad pins.
- Memory location and storage element are used interchangeably and both refer to 1 byte (8 bits).

4.4.1 Space Complexity

The algorithm stores the template for each different FPGA block. Since the detailed router needs more detailed structure, the algorithm uses different data structure than the global router. The data structure used to store this information is shown in Figure 76. As shown in the figure,

1. Each UPM needs PPS*4+PT memory locations.
2. Each SB needs P*F memory locations to store the required information.
3. Each CB needs PT*2+1 memory locations for each track. For MT tracks, MT(2PT+1) locations are required.
4. The I/O pad needs IOP locations.

The detailed router needs to store information for 24 templates for single UPM FPGA

(Figure 32,page 47) or 28 templates for back-to-back (Figure 33, page 48).

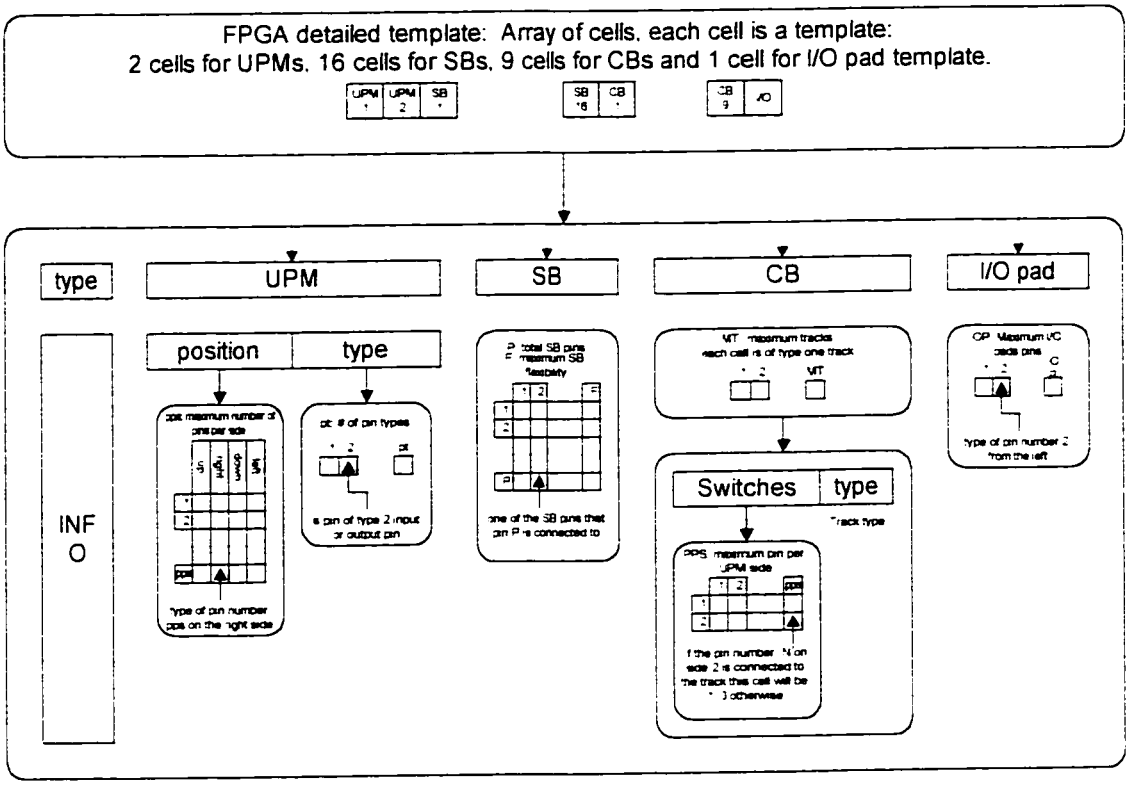


Figure 76: Detailed template data structure.

The detailed router needs to store the solution tree for each net. Only the solution tree of the current net needs to be stored. The tree needs ML locations to store each branch. With a maximum of M branches, a total of M*ML storage location are required.

The FPGA is modeled by a 2-D array of size equal to (R+2)*(C+2) (two columns and two rows are added for the I/O pads) (Figure 77). Each cell in the array is pointing to some structure containing information about the cell. This structure is used to store

information regarding the number of paths using each block. A cell type field is used to classify the cell into one of the available types (24 types for single UPM configuration and 28 for back-to-back). As the algorithm proceeds, the resources are updated and the resources fields are adjusted accordingly.

The cell type requires one locations for each cell (in total (R+1)(C+1)= RC+R+C+2). Storage space required by cell template depends on the cell type. The UPM requires 4PPS locations, the SB requires P, the CB requires MT and the I/O pad requires IOP memory locations. A single UPM configuration with R rows and C columns has:

- $\left(\frac{R-1}{2}\right)\left(\frac{C-1}{2}\right)=\frac{RC-R-C+1}{4}$ UPMs,
- $\left(\frac{R+1}{2}\right)\left(\frac{C+1}{2}\right)=\frac{RC+R+C+1}{4}$ SBs,
- $\left(\frac{R-1}{2}\right)\left(\frac{C+1}{2}\right)=\frac{RC+R-C-1}{4}$ Vertical CBs,
- $\left(\frac{R+1}{2}\right)\left(\frac{C-1}{2}\right)=\frac{RC-R+C-1}{4}$ Horizontal CBs, and
- $2(R+C)$ I/O pads

Alternately, the back-to-back configuration with R rows and C columns has:

- $\left(\frac{R-1}{2}\right)\left(2\frac{C-1}{3}\right)=\frac{2RC-2R-2C+2}{6}$ UPMs,
- $\left(\frac{R+1}{2}\right)\left(\frac{C+2}{3}\right)=\frac{RC+2R+C+2}{6}$ SBs,
- $\left(\frac{R-1}{2}\right)\left(\frac{C+2}{3}\right)=\frac{RC+2R-C-2}{6}$ Vertical CBs,

- $\left(\frac{R+1}{2}\right)\left(2\frac{C-1}{3}\right)=\frac{2RC-2R+2C-2}{6}$ Horizontal CBs , and
- $2(R+C)$ I/O pads

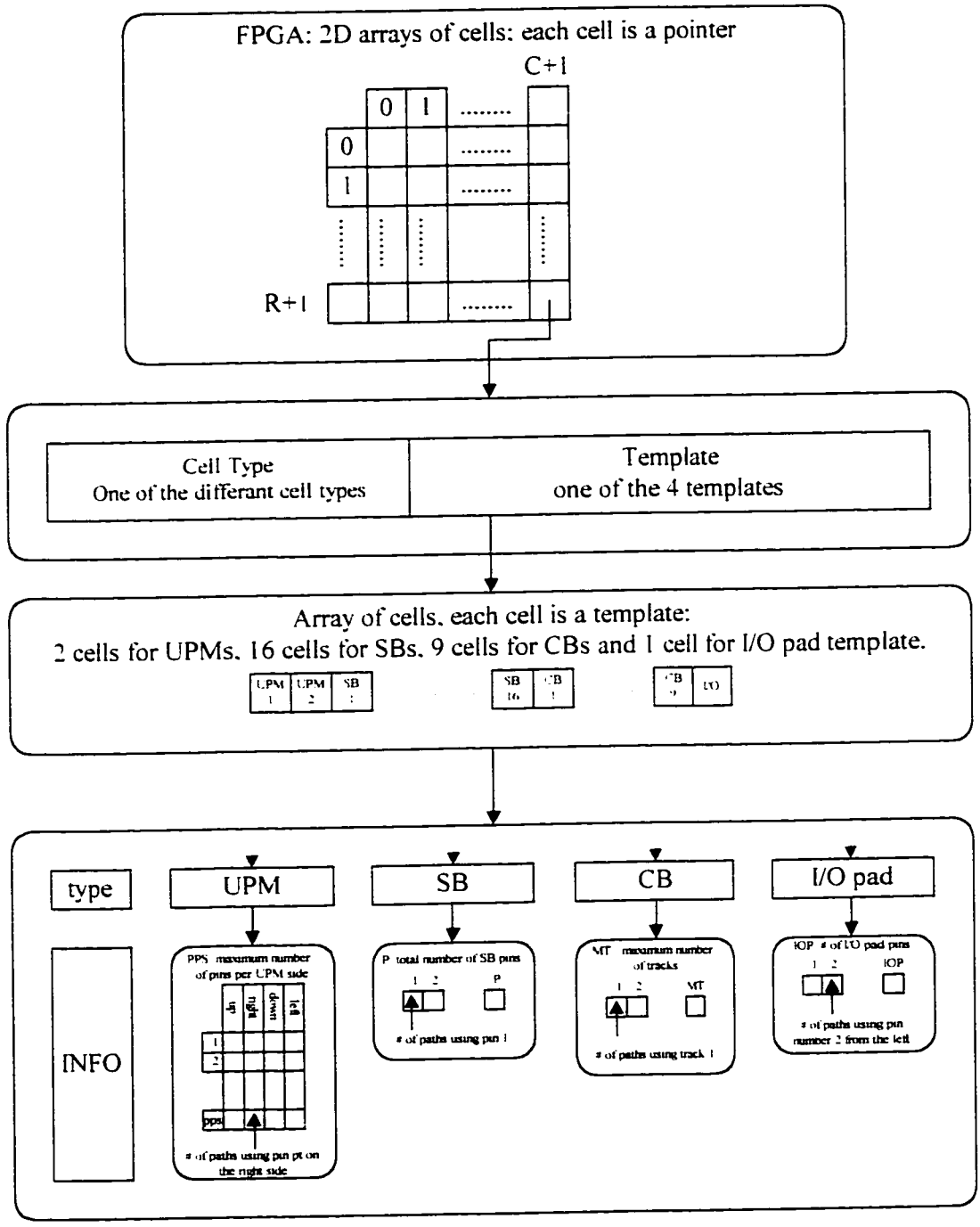


Figure 77: FPGA array data structure.

During the expansion step, the algorithm needs to store the expansion of the current path. This requires three storage locations for each cell in the path, two for the cell coordinates and one to indicate the resources used in the current expansion. This is a total of 3*ML locations.

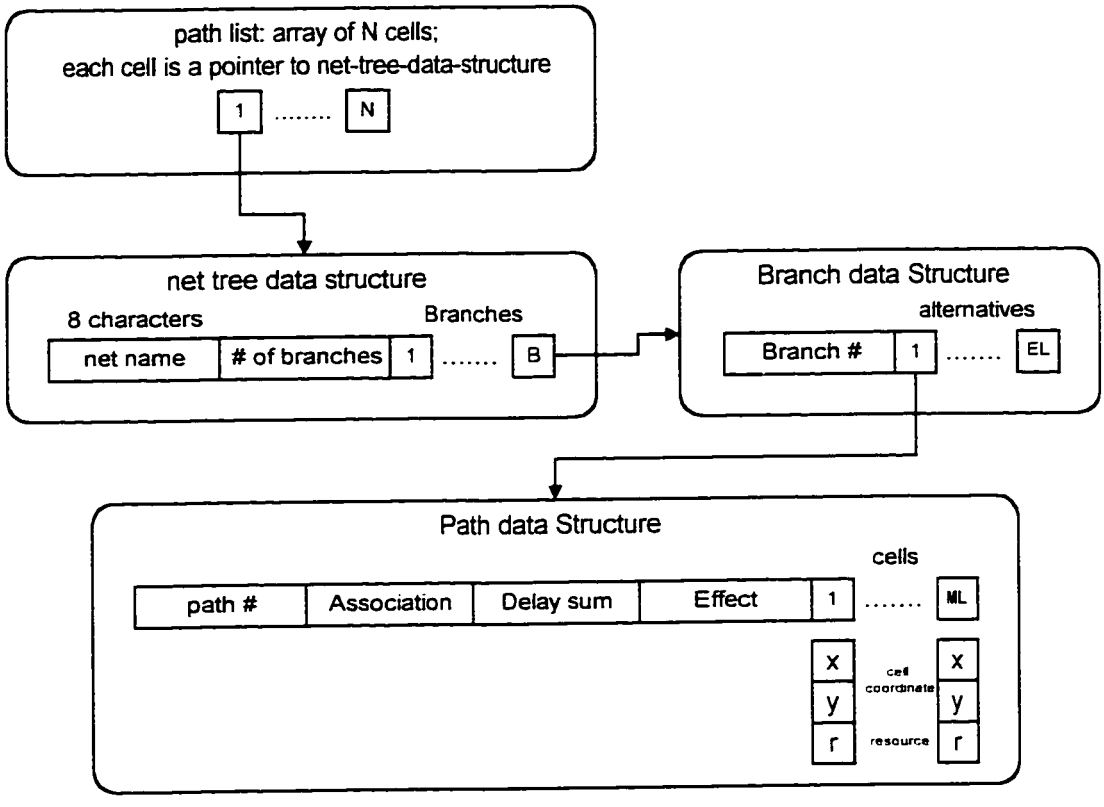


Figure 78: Path-list data structure.

The path list constructed on the expansion phase needs to be stored. The data structure for this list is shown in Figure 78. It requires 3*ML+4 for each branch alternative. For EL alternatives, the required storage is EL*(3ML+4) for each branch. Thus, each net requires 2+B*EL*(3ML+4) locations. The path list requires

N(2+B*EL*(3ML+4)) locations or simply O(N*EL*ML) locations. Since ML upper bound is 2(R+C) then the space requirements can be written as O(N*EL*(R+C)).

The connection formation phase selects one of these alternatives for each path and returns the required data (no need for storage).

From all of the above, since all values other than N, ML, R and C are relatively small, the space requirements of the detailed router is of O(N*ML+RC). Since ML upper bound is 2(R+C) then the space requirements can be written as O(N*(R+C)+RC).

4.4.2 Time Complexity

In the expansion phase, the detailed router performs O(ML) operations[1] for each expansion. The number of expansions for each path is limited by EL. Therefore, the total number of operations for expanding all nets is O(N*M*ML*EL) or O(N*M*(R+C)*EL). To compute the cost, the router needs to visit all cells on each path on the path list. This requires O(N*M*(R+C)*EL) operations.

During the connection formation phase, the algorithm searches the path-list for the lowest cost path. While selecting the first connection, the path-list size is O(N*M*(R+C)*EL). After each connection, the path-list is searched to remove all

[1] Operation refers to either a logical or arithmetical operation

paths conflicting with the new connection and update the costs of the remaining paths. In the following selection, the path-list size is reduced. At least all alternative paths for the current connected net are removed. Thus, the upper bound of the path-list size is O(N*M*(R+C)*EL). Therefore, connecting all nets requires $O(N^2*M*(R+C)*EL)$ or simply $O(N^2*(R+C))$ since M and EL are relatively small.

From the previous discussion, the detailed router time complexity is of $O((R+C)(N^2+N))$ or simply $O((R+C)N^2)$.

CHAPTER 5

EXPERIMENTAL RESULTS &

DISCUSSION

The developed global and detailed routing algorithms were tested and fine-tuned using a number of locally developed test circuits. The templates of these test circuits were manually constructed and expressed in the required format. Another set of test circuits was developed to test the router performance using fuzzy and non-fuzzy cost evaluation techniques. Furthermore, two fuzzy logic operators: the *ordered weighted average* (OWA) and the additive combiner were tested.

In the following section, the performance-evaluation test circuits are described. The used different cost functions together with the obtained results are given. The effect of using different shapes of the membership function on the solution quality of the additive combiner fuzzy logic operator is also discussed.

5.1 Test Circuits

Mapping a given design onto a specific FPGA platform passes through several phases. The routing algorithm, the final step in such mapping, receives its design netlist input from the placement phase which follows the technology-mapping step. At the time this work was being conducted, the placement tool was not yet ready. Accordingly, the test circuits were placed manually. The full description of the templates of all blocks of the used FPGA is given in APPENDIX 1.

The first test circuit is a 13 nets circuit placed on a 7*7 FPGA. This FPGA has 9 UPMs and 7 I/O pads on each side. The smallest net in this circuit has two pins and the largest has 5 pins. The average net size is 3.077 (APPENDIX 2).

The second test circuit is a 26 net circuit placed on 21*21 FPGA. This FPGA has 100 UPMs and 21 I/O pads on each side. Nets in this circuit vary from 2 to 7 pins. The average net size is 3.154 (APPENDIX 2).

For the same FPGA, bigger circuits with 60 and 80 nets were placed and two more test circuits were developed. Nets in both circuits vary from 2 to 8 pins. The average net size is 3.083 and 3.050 respectively (APPENDIX 2).

5.2 Experimental Results

Results for the 4 test circuits are tabulated next. Each table shows the test results for the tested techniques. The following parameters are used as basis of comparison:

- **Routability**: is the percentage of routed nets over the total number of nets.
- **Average Tree Size:** is the sum of the sizes of various solution trees of all routed nets over the number of routed nets.
- **Average Number of Turns:** is the total number of turns in all solution trees of all routed nets over the number of routed nets.
- The row labeled *"Minimum"* shows average tree size and the average number of turns obtained by running the router assuming an FPGA with infinite routing resources.

5.2.1 Different Cost Evaluation Techniques

The first set of tests was performed to see the effect of using different cost evaluation techniques on the solution quality. The first tested cost function uses a greedy approach where the components of the cost function are added up and the solution with the minimum sum is selected. The second tested technique is the weighted sum technique in which each component of the cost function is given some weight W, and the total cost is calculated as follows:

$$\text{Total Cost} = \sum_{\forall \text{cost elements } i} W_i \text{Cost}_i$$

The solution with the minimum total weighted cost is selected.

Two different fuzzy operators were used and their results were compared with the greedy and the weighted sum techniques. These are the additive combiner and the OWA (section 3.2.6). All test results for the 4 test circuits are shown in Table 10. The average tree size and average number of turns have been normalized to the minimum value and plotted in Figure 79 and Figure 80 respectively.

Table 10: Test Results for Different Cost Evaluation Techniques.

	Array Size	7*7	21*21	21*21	21*21
	Number of nets	13	26	60	80
	Smallest net	2	2	2	2
	Average net size	3.077	3.154	3.083	3.050
	Largest net	5	7	8	8
Minimum	Average tree size	8.538	10.385	17.483	17.463
	Average number of turns	3.692	4.731	6.950	6.925
Greedy	Routability	100%	100%	100%	97%
	Average tree size	9.846	11.885	18.933	20.692
	Average number of turns	4.923	6.038	8.300	9.372
Weighted sum	Routability	100%	100%	98%	98%
	Average tree size	9.385	11.769	18.881	20.375
	Average number of turns	4.385	5.692	8.183	9.253
Fuzzy Logic additive combiner	Routability	100%	100%	100%	100%
	Average tree size	9.077	11.538	18.550	20.475
	Average number of turns	4.231	5.577	7.900	8.950
Fuzzy Logic OWA with β=0.6	Routability	100%	100%	100%	96%
	Average tree size	9.308	11.346	18.950	21.494
	Average number of turns	4.385	5.615	8.083	9.506

The results show that the fuzzy logic additive combiner operator gives better results (closer to the minimum) in almost all the cases. In addition, this technique is the only one that achieved 100% routability for the four test circuits.

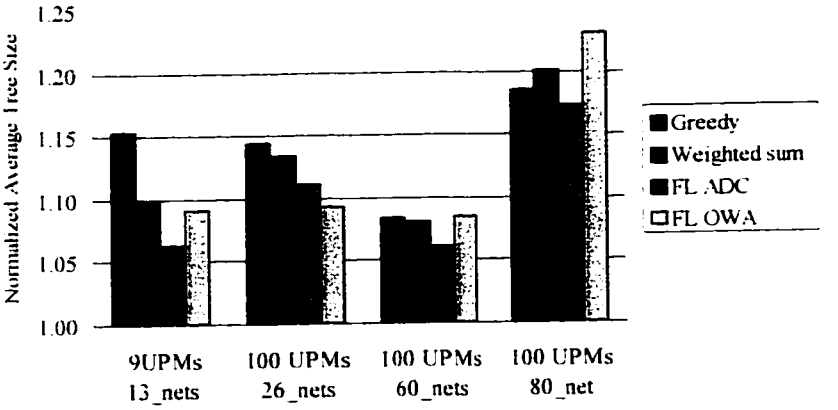


Figure 79: Normalized average tree size of the test results for different cost evaluation techniques.

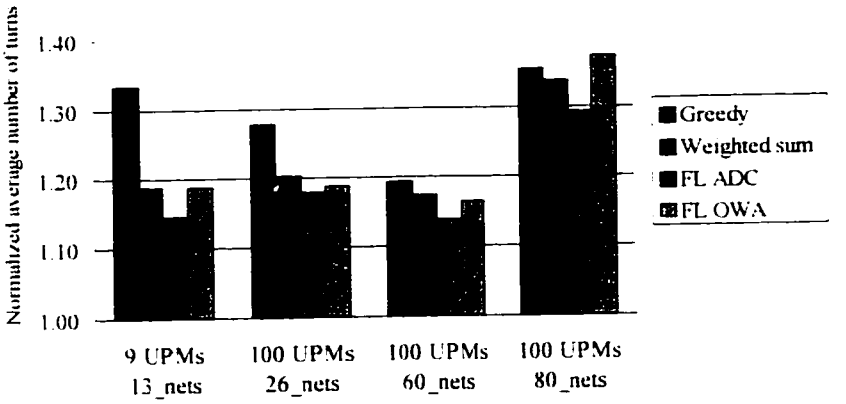


Figure 80: Normalized number of turns of the test results for different cost evaluation techniques.

5.2.2 Varying OWA parameter

The second set of tests was performed by varying β of the OWA fuzzy operator (section 3.2.6). Test results for the 4 test circuits are shown in Table 11. The average

tree size and average number of turns have been normalized to the minimum value and plotted in Figure 81 and Figure 82 respectively.

Table 11: Test Results for Varying β of The Fuzzy Operator OWA.

	Array Size	7*7	21*21	21*21	21*21
	Number of nets	13	26	60	80
	Smallest net	2	2	2	2
	Average net size	3.077	3.154	3.083	3.050
	Largest net	5	7	8	8
Minimum	Average tree size	8.615	10.385	17.450	17.450
	Average number of turns	3.692	4.731	6.950	6.925
$\beta = 0$	Routability	100%	100%	100%	100%
	Average tree size	9.077	11.538	18.500	20.563
	Average number of turns	4.231	5.577	7.850	8.913
$\beta = 0.2$	Routability	100%	100%	100%	100%
	Average tree size	9.231	11.346	18.617	20.838
	Average number of turns	4.385	5.615	7.983	9.200
$\beta = 0.6$	Routability	100%	100%	100%	96%
	Average tree size	9.308	11.346	18.950	21.494
	Average number of turns	4.385	5.615	8.083	9.506
$\beta = 0.8$	Routability	100%	100%	100%	100%
	Average tree size	9.385	11.500	18.983	20.838
	Average number of turns	4.462	5.654	8.083	9.150
$\beta = 1$	Routability	100%	100%	98%	99%
	Average tree size	9.692	11.346	19.186	21.823
	Average number of turns	4.615	5.808	8.508	9.975

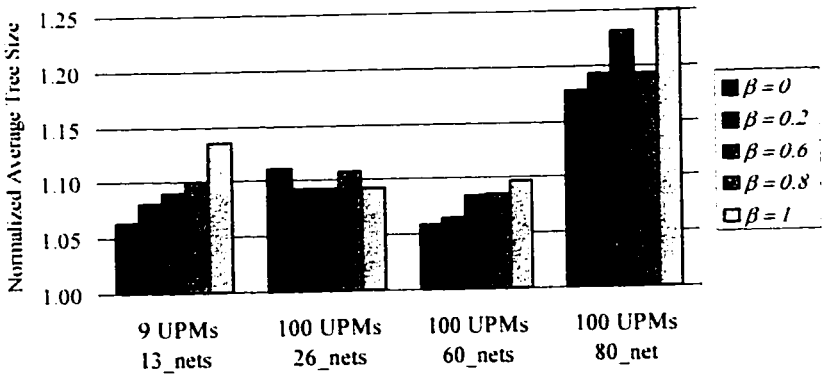


Figure 81: Normalized average tree size of test results for varying β of the fuzzy operator OWA.

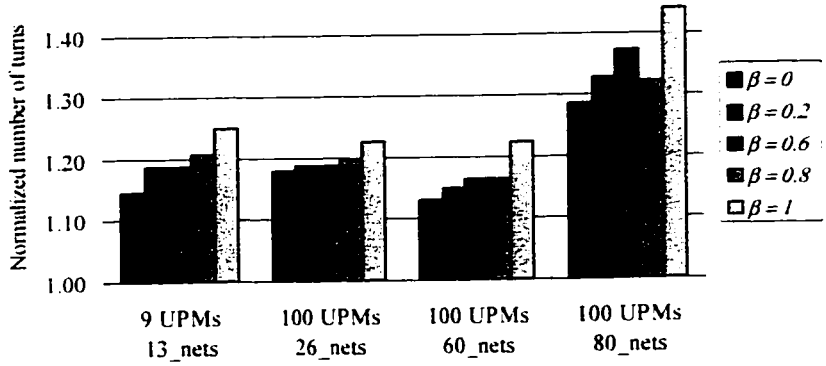


Figure 82: Normalized average number of turns of test results for varying β of the fuzzy operator OWA.

As it can be seen from the results, the average operator ($\beta = 0$) gives better results. For some values of β the router could not route all the nets on some test circuits ($\beta = 0.6$ and $\beta = 1$).

5.2.3 Different Shapes of Membership Function

The third set of tests was performed by varying the shapes of the membership function and using the additive combiner fuzzy operator to combine the various cost function elements (section 3.2.6). The different membership function shapes are shown in Figure 83. Test results for the 4 test circuits are shown in Table 12. The normalized average tree size and average number of turns are plotted in Figure 84 and

Figure 85 respectively. The results show that the membership function with two slopes yields better solutions. The step and step/slope shapes made the router unable to route all nets in some test circuits.

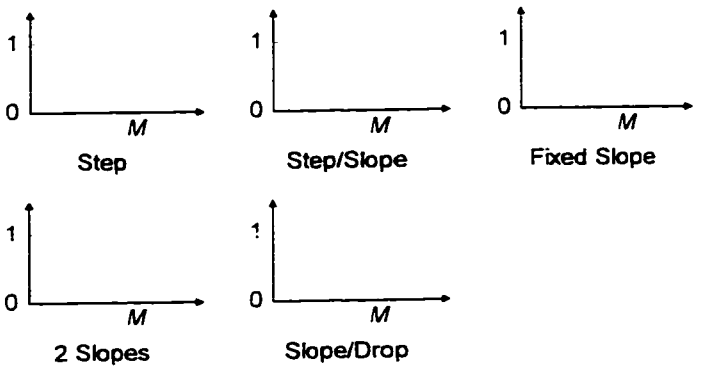


Figure 83: Different shapes of membership function.

Table 12: Test Results for Using Different Shapes of Membership Function.

	Array Size	7*7	21*21	21*21	21*21
	Number of nets	13	26	60	80
	Smallest net	2	2	2	2
	Average net size	3.077	3.154	3.083	3.050
	Largest net	5	7	8	8
Minimum	Average tree size	8.615	10.385	17.450	17.450
	Average number of turns	3.692	4.731	6.350	6.325
Step	Routability	100%	100%	100%	98%
	Average tree size	9.846	11.885	19.317	21.179
	Average number of turns	4.923	6.038	8.383	9.603
Step/slope	Routability	100%	100%	100%	99%
	Average tree size	9.923	11.885	18.983	21.278
	Average number of turns	4.923	6.038	8.450	9.747
Fixed slope	Routability	100%	100%	100%	100%
	Average tree size	9.077	11.577	18.550	20.525
	Average number of turns	4.308	5.615	7.850	9.000
2 slopes	Routability	100%	100%	100%	100%
	Average tree size	9.077	11.538	18.550	20.475
	Average number of turns	4.231	5.577	7.900	8.950
slope/drop	Routability	100%	100%	100%	100%
	Average tree size	9.385	11.577	18.567	20.425
	Average number of turns	4.385	5.615	7.933	8.888

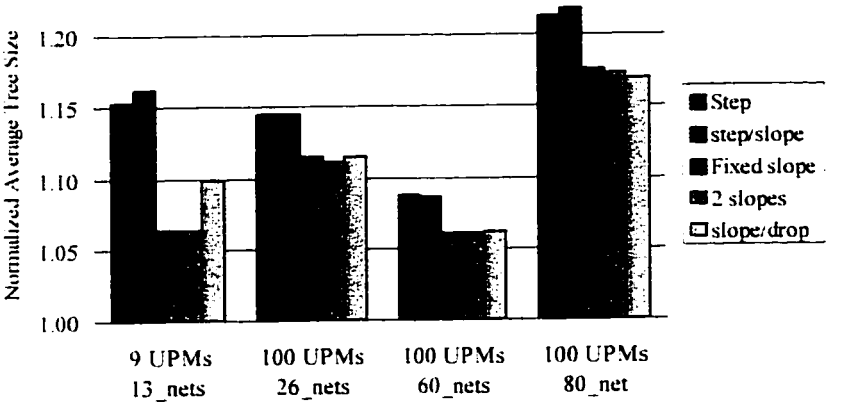


Figure 84: Normalized average tree size of the test results for using different shapes of membership function.

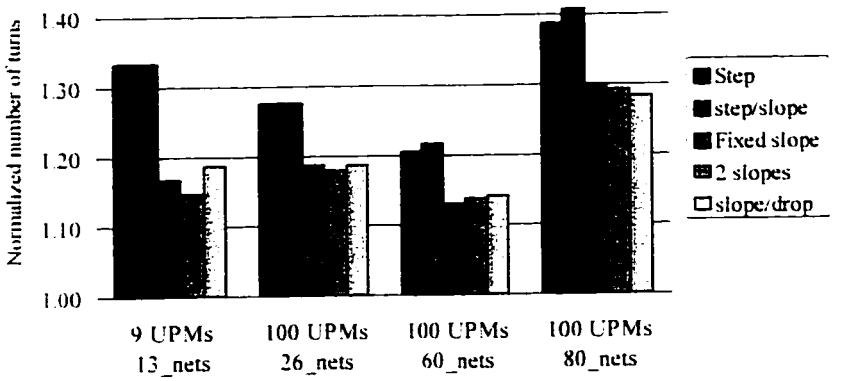


Figure 85: Normalized average number of turns of the test results for using different shapes of membership function.

5.3 Other Results

A number of other results related to FPGA architecture and the routing algorithm have been observed. There are listed below.

- The distribution of switches in the switch boxes generally affects routability. This result was observed by running a number of tests on the same circuit but with different switch boxes templates.
- Although typical connection box flexibility is 80% of the channel width, the router achieves 100% connectivity for some test circuits with connection box flexibility as low as 40%. This is because the adopted UPM has its output available on three sides and its logically equivalent input pins are available on the three sides as well.
- The net order affects the total wire length.
- The program run-time is very small, varying between few seconds for small FPGAs to few minuets for large FPGAs. Accordingly, no special runtime reduction techniques were used.
- Giving the congestion cost lower weight than the other two components in the total cost function usually leads to better results. This gives more chance in selecting more congested paths which leads to more space (the lightly congested cells) to be used in routing the yet unconnected nets.

CHAPTER 6

CONCLUSION & FUTURE WORK

The last two decades have witnessed a tremendous increase in the demand for application specific VLSI chips (ASICs). ASICs with fast TAT have been particularly attractive for the early introduction of digital systems into the market. Automated design tools help speed up the design process. FPGAs have been gaining popularity in this regard due to their fast TAT and possible reprogrammability.

In this thesis, an automated fuzzy router for a new family of island-based FPGAs has been developed. Typically, automated routing tools for FPGAs are device specific since they have to deal with a pre-specified set of routing segments and structure.

Although the router is designed for a specific FPGA architecture, most included features were designed general enough to render it flexible for a variety of architectural parameters. Island-based FPGAs have four main structures: UPMs, switch boxes, connection boxes and I/O pads.

The router developed in this work allows:

- Either single or back-to-back UPM configuration.
- Various FPGA array sizes.
- Flexible UPM configuration regarding the pin types and number on any side. It can also handle UPMs which allow one or more pins to appear on more than one side.
- Flexible I/O pads configuration regarding the pin types and numbers on any side.
- Flexible architectures of various routing resources (switch boxes and connection boxes programmable switches, and routing channels tracks and their segments).

All different types of modules might have a number of variations. The different types and their template formats are described in CHAPTER 2

The routing problem is usually divided into two major steps: global and detailed routing. The global routing, discussed in CHAPTER 3, is implemented using a variation of Lee maze routing algorithm. The global router produces a coarse Steiner tree for each net. It builds the Steiner tree one branch at a time where a branch is a path established between either two net pins or between a net pin and a Steiner point on the tree.

The router uses a fuzzified cost function. The cost function is designed to minimize the total path delay and block congestion. Furthermore, preference is given to route nets through blocks within the smallest rectangle that binds all net pins.

The space complexity of the global router is O(u), where u is the number of logic modules in the FPGA which is directly proportional to R*C and N. R and C are the number of FPGA rows and columns respectively and N is number of nets in the design. The time complexity is $O(u^2)$.

The detailed router takes as input the coarse graphs for all nets, provided by the global router, and produces the set of wire segments and switches to be used for establishing the connection of these nets. The detailed router is based on the coarse graph expansion (CGE) algorithm. The cost function, which is developed to minimize the total path delay while connecting all nets, is fuzzified to improve the algorithm performance.

The space complexity of the detailed router is O(u) and its time complexity is $O(u^2)$. An expansion limit factor is used to limit the expansion procedure to reduce the algorithm complexity while still providing enough choices to route all nets.

Both the global and the detailed routers have been implemented and tested using locally developed test circuits. The results were very promising. However, more thorough testing is required. The router can adopt any future adjustment to the FPGA architecture as long as it maintains similar architecture. Although the router is designed for a specific FPGA architecture, with minimum adjustments, it can be used with other FPGAs.

FUTURE WORK

- Studying the effect of other parameters of the cost function on solution quality.
- Fine tuning of the cost function for better solution quality.
- Investigation of other routing algorithms.
- Incorporating a rip-up and reroute feature.

APPENDIX 1: FPGA TEMPLATES

Template of the Universal Programmable Module[1]

```
1 1 2 3 0     up
2 1 4 0       right
5 1 2 1 0     down
0             left
-2      1 3 4 5 0
-1      2 0
```

Template of connection boxes[2]

The internal Horizontal connection box

```
 1 1 2 0 1 2       -1
 2 1 2 0 1 2       -4
 3 3 4 0 3 4       -4
 4 1 3 0 2 4       -2
 5 1 3 0 2 4       -3
 6 2 4 0 1 3       -3
 7 2 4 0 1 3       -2
 8 1 2 0 3 4       -4
 9 3 4 0 1 2       -4
10 3 4 0 3 4       -1
0
```

The internal Vertical connection box

```
 1 1 2 0 1 2       -1
 2 1 2 3 0 1 2     -4
 3 1 2 3 0 1 2 3   -2
 4 1 3 0 1 2       -4
 5 3 0 3           -1
 6 1 2 0 1 2       -3
 7 1 3 0 1 3       -4
 8 2 0 2 3         -4
0
```

[1] See section 2.2.2.1: The Pinout of Universal Programmable Modules on page 35

[2] See section 2.2.2.3: Connection Boxes on page 37

The peripheral horizontal upper connection box

```
 1  1  3  3  0  1  3      -4
 2  2  4  4  0  2  4      -4
 3  3  1  0  3  2  4      -4
 4  4  2  0  4  2         -4
 5  1  3  0  3  2         -2
 6  2  4  0  2  4         -3
 7  3  1  0  1  1         -2
 8  4  2  0  2  2         -3
 9  1  3  0  3  3         -2
10  2  4  0  4  3         -3
0
```

The peripheral horizontal lower connection box

```
 1  1  3  3  0  1  3      -2
 2  2  4  4  0  2  4      -3
 3  3  1  0  3  2  4      -2
 4  4  2  0  4  2         -3
 5  1  3  0  3  2         -2
 6  2  4  0  2  4         -3
 7  3  1  0  1  1         -4
 8  4  2  0  2  2         -4
 9  1  3  0  3  3         -4
10  2  4  0  4  3         -4
0
```

The peripheral vertical right connection box

```
1  2  4  0  3            -4
2  1  2  0  1  2  4       -4
3  1  2  3  0  1  2  3    -4
4  1  2  3  0  1  2  3    -4
5  1  2  0  1  2  3       -3
6  2  3  4  0  1  4       -3
7  1  4  0  1  2  4       -2
8  1  3  4  0  4          -2
0
```

The peripheral vertical left connection box

```
1  1  2  0  1  2  3       -2
2  2  3  0  1             -2
3  1  2  0  1  2          -3
4  1  3  4  0  4          -3
5  2  0  3                -4
6  1  2  0  1  2          -4
7  2  0  3                -4
8  1  2  0  1  2          -4
0
```

Template of switch boxes[1]

Even-Even Switch box

```
2  10  15  19  28  0    on top
3  11  20  29  33  0
4  12  23  30  34  0
7  17  19  24  35  0
8  16  25  28  33  0
10  19  28  0          on right
11  20  29  0
12  23  30  0
15  25  34  0
16  19  24  35  0
17  20  28  0
19  0           on bottom
20  29  35  0
23  30  0
24  33  0
25  34  0
0
```

Even-Odd switch box

```
2  10  19  28  0        on top
11  20  24  29  0
4  13  23  31  0
7  17  19  25  35  0
8  10  16  28  0
10  19  28  0          on right
11  20  29  31  0
13  24  34  35  0
14  23  29  31  0
16  25  32  34  0
17  20  28  32  0
19  0           on bottom
20  29  0
23  32  0
24  35  0
25  34  0
0
```

[1] see section 2.2.2.4:Switch Boxes on page 40

Odd-Even switch box

```
2 10 19 28 0        on top
4 12 23 30 0
6 16 25 34 0
7 11 17 19 35 0
8 10 12 19 28 0
10 19 28 0          on right
11 20 29 0
12 23 35 0
15 21 25 33 0
16 21 35 0
17 20 28 0
19 0          on bottom
20 29 33 0
21 33 0
23 30 34 0
25 34 0
0
```

Odd-Odd switch box

```
2 10 19 28 0        on top
4 13 23 29 0
6 16 21 25 34 0
7 17 19 35 0
8 10 16 28 32 0
10 19 28 32 0       on right
11 20 29 32 0
13 21 31 0
14 23 31 34 0
16 19 31 0
17 20 25 28 0
19 0          on bottom
20 29 35 0
21 32 0
23 35 0
25 34 0
0
```

Peripheral upper even switch box

```
1 9 21 23 32 0     on top
2 10 19 25 28 0
3 11 20 35 0
4 13 23 30 0
9 22 36 0          on right
10 19 28 0
11 20 30 0
12 26 32 35 0
13 24 26 33 0
15 21 22 32 0
17 20 28 34 0
19 28 0            on bottom
20 36 0
21 32 0
22 36 0
23 30 0
24 33 30 0
25 34 33 0
26 32 0
0
```

Peripheral upper odd switch box

```
1 9 21 24 27 0     on top
2 10 19 33 0
3 11 20 29 0
4 12 23 34 0
9 22 31 0          on right
10 19 25 27 0
11 20 29 0
12 21 31 0
14 26 27 36 0
16 19 25 35 0
18 21 27 35 0
19 22 31 0         on bottom
20 29 0
21 36 0
22 35 0
23 30 33 0
24 33 34 0
25 34 0
26 36 0
0
```

Peripheral Left even switch box

```
1  9  21  33  0          on top
2  10  19  34  0
5  14  22  33  0
6  11  16  25  34  0
7  12  17  19  35  0
8  10  16  35  0
9  22  26  0             on right
10  19  0
11  20  22  0
12  20  22  0
13  21  33  0
14  22  35  0
15  25  36  0
16  19  35  0
17  20  0
18  25  36  0
19  0            on bottom
20  0
21  0
22  0
25  34  0
26  36  0
0
```

Peripheral left Odd switch box

```
3 11 20 33 0        on top
4 13 23 34 0
5 14 24 33 0
6 16 21 34 0
7 9 17 19 35 0
8 10 16 35 0
9 22 36 0           on right
10 19 24 0
11 20 33 0
12 21 34 0
13 24 33 0
14 22 36 0
15 23 36 0
16 19 35 0
17 20 24 0
18 21 22 0
19 0          on bottom
20 0
21 0
22 33 0
23 34 0
24 35 0
0
```

Peripheral down even witch box

```
1 9 23 27 0         on top
2 15 25 28 0
3 11 26 29 0
4 13 23 30 0
5 14 24 33 0
6 16 25 34 0
7 17 23 32 0
8 16 25 28 0
9 26 36 0           on right
11 26 29 0
13 24 27 0
15 30 34 0
16 32 0
17 28 32 0
18 29 36 0
23 30 34 0          on bottom
24 36 0
25 34 0
26 30 0
0
```

Peripheral down Odd switch box

```
1 12 23 27 0        on top
2 10 24 28 0
3 14 25 29 0
4 15 26 23 30 0
5 14 24 33 0
6 16 25 27 0
7 17 25 35 0
8 10 16 26 28 0
10 28 0             on right
12 26 31 0
14 23 30 0
15 27 0
16 35 0
17 24 28 0
18 26 31 33 0
23 30 0             on bottom
24 33 0
25 29 35 0
26 27 31 0
0
```

Peripheral right even switch box

```
1 9 25 27 33 0      on top
2 10 19 28 34 0
3 11 20 29 0
4 12 23 30 0
7 12 19 35 0
8 10 26 28 0
9 26 31 35 0        on right
10 19 28 0
11 20 29 0
12 23 32 36 0
19 28 0             on bottom
20 29 0
23 30 0
24 31 33 0
25 32 34 0
26 36 0
0
```

Peripheral right Odd switch box

```
1  9  21  27  0          on top
2  10  23  28  29  0
3  11  24  29  30  0
4  12  23  30  31  0
5  12  24  32  33  0
6  11  25  33  34  0
9  22  31  34  0         on right
10  22  28  35  0
11  26  29  36  0
12  21  32  0
21  27  0               on bottom
22  31  0
23  30  0
24  33  0
25  27  34  0
26  28  36  0
0
```

Upper left Corner switch box

```
1  9  13  22  26  0      on top
2  10  14  21  25  0
3  11  15  20  24  0
4  12  16  19  23  0
9  19  26  33  0         on right
10  20  25  34  0
11  21  24  35  0
12  22  23  36  0
13  19  23  36  0
14  20  24  0
15  21  25  36  0
16  22  26  0
17  34  0
18  33  0
19  36  0               on bottom
20  35  0
21  34  0
22  33  0
23  33  0
24  35  0
25  34  0
26  36  0
0
```

Upper right Corner switch box

```
1 22 26 27 23 0   on top
2 21 25 28 34 0
3 20 24 29 35 0
4 19 23 30 36 0
9 19 23 30 36 0   on right
10 20 24 31 35 0
11 21 25 30 34 0
12 22 26 29 33 0
19 27 30 0        on bottom
20 28 31 0
21 29 32 0
22 33 36 0
23 34 35 0
24 27 36 0
25 28 35 0
26 29 34 0
0
```

Lower left Corner switch box

```
1 9 18 23 33 0    on top
2 10 17 24 34 0
3 11 16 25 35 0
4 12 15 26 36 0
5 9 14 26 33 0
6 10 15 25 34 0
7 11 16 24 35 0
8 12 17 23 36 0
9 23 36 0         on right
10 24 35 0
11 25 34 0
12 26 33 0
13 24 33 0
14 23 33 0
15 24 34 0
16 26 35 0
17 25 36 0
18 0
23 0          on bottom
24 0
25 0
26 0
0
```

Lower right Corner switch box

```
1 9 23 27 36 0     on top
2 10 24 28 35 0
3 11 25 29 34 0
4 12 26 30 33 0
5 9 26 31 32 0
6 10 25 32 36 0
7 11 24 33 35 0
8 12 23 34 0
9 27 36 0          on right
10 28 35 0
11 29 34 0
12 30 33 0
23 31 32 0         on bottom
24 32 36 0
25 33 35 0
26 34 27 0
0
```

I/O pad template description[1]

```
-1 -2 -3 -4 0      I/O PAD
```

[1] see section 2.2.2.5: I/O Pads on page 43

APPENDIX 2: NETLIST OF TEST CIRCUITS

13 nets on 7*7 FPGA (9 UPMs)

```
k       2 2 2 2 4 1 -1 27
k2      2 4 2 2 6 1 4 2 1 0
i       2 6 2 4 2 1 -1 0
n2      4 2 2 4 4 1 4 6 1 6 4 1 0
j       4 4 2 4 6 1 6 4 1 -1 0
u5      4 6 2 6 4 1 0
h       6 4 2 -1 0
c       4 4 1 4 6 1 6 4 1 -2 0
b       2 6 1 4 2 1 -2 0
a       2 2 1 2 4 1 -2 0
g       2 2 1 2 4 1 -2 0
d       2 2 1 2 4 1 -2 0
e       2 6 1 4 2 1 -2 0
END.
```

26 nets on 21*21 FPGA (100 UPMs)

```
n001    2 2 2 2 4 1 -1 80
n002    8 6 2 2 8 5 2 6 1 0
n003    2 4 2 2 6 1 4 2 1 0
n004    2 6 2 4 2 1 -1 0
n005    4 2 2 4 4 1 4 6 1 6 4 1 0
n006    4 4 2 4 6 1 6 4 1 -1 0
n007    4 6 2 6 4 1 0
n008    6 4 2 -1 0
n009    4 4 1 4 6 1 6 4 1 -2 0
n010    2 6 1 4 2 1 -2 0
n011    2 2 1 2 4 1 -2 0
n012    2 2 1 2 4 1 -2 0
n013    2 2 1 2 4 1 -2 0
n014    2 6 1 4 2 1 -2 0
n015    2 8 2 8 2 1 8 8 1 8 6 1 0
n016    4 8 2 4 2 1 4 4 1 4 6 1 6 4 1 6 6 1 8 6 1 8 8 1 0
n017    6 2 2 -1 0
n018    6 2 1 8 4 1 -2 0
n019    6 6 2 6 8 3 0
n020    6 8 2 6 4 4 0
n021    8 2 2 8 6 4 -1 0
n022    8 4 2 2 8 4 -1 0
n023    8 8 2 6 8 1 6 6 3 -1 0
n024    8 8 1 8 6 1 -2 0
n025    2 8 1 4 8 1 -2 0
n026    6 2 1 -2 0
END.
```

60 nets on 21*21 FPGA (100 UPMs)

```
n001    2 2 2 2 14 1 -1 80
n002    8 6 2 2 18 5 2 6 1 0
n003    2 4 2 2 16 1 4 2 1 0
n004    2 6 2 4 12 1 -1 0
n005    4 2 2 4 14 1 4 6 1 6 4 1 0
n006    4 4 2 4 16 1 6 4 1 -1 0
n007    4 6 2 6 14 1 0
n008    6 4 2 -1 0
n009    4 4 1 4 16 1 6 4 1 -2 0
n010    2 6 1 4 12 1 -2 0
n011    2 2 1 2 14 1 -2 0
n012    2 2 1 2 14 1 -2 0
n013    2 2 1 2 14 1 -2 0
n014    2 6 1 4 12 1 -2 0
n015    2 8 2 8 12 1 8 8 1 8 6 1 0
n016    4 8 2 4 12 1 4 4 1 4 6 1 6 4 1 6 6 1 8 6 1 8 8 1 0
n017    6 2 2 -1 0
n018    6 2 1 8 14 1 -2 0
n019    6 6 2 6 18 3 0
n020    6 8 2 6 14 4 0
n021    8 2 2 8 16 4 -1 0
n022    8 4 2 2 18 4 -1 0
n023    8 8 2 6 18 1 6 6 3 -1 0
n024    8 8 1 8 16 1 -2 0
n025    2 8 1 4 18 1 -2 0
n026    6 2 1 -2 0
n027    10 10 2 10 2 1 10 4 1 0
n028    10 8 2 10 2 1 10 6 1 0
n029    10 6 2 10 2 1 10 10 1 0
n030    10 4 2 10 2 1 10 20 1 0
n031    12 12 2 12 4 1 -1 81
n032    18 16 2 12 8 5 12 16 1 0
n033    12 14 2 12 6 1 14 12 1 0
n034    12 16 2 14 2 1 -1 0
n035    14 12 2 14 4 1 14 16 1 16 14 1 0
n036    14 14 2 14 6 1 16 14 1 -1 0
n037    14 16 2 16 4 1 0
n038    16 14 2 -1 0
n039    14 14 1 14 6 1 16 14 1 -2 0
n040    12 16 1 14 2 1 -2 0
n041    12 12 1 12 4 1 -2 0
n042    12 12 1 12 4 1 -2 0
n043    12 12 1 12 4 1 -2 0
n044    12 16 1 14 2 1 -2 0
```

```
n045    12 18 2 18 2 1 18 18 1 18 16 1 0
n046    14 18 2 14 2 1 14 14 1 14 16 1 16 14 1 16 16 1 18
        16 1 18 18 1 0
n047    16 12 2 -1 0
n048    16 12 1 18 4 1 -2 0
n049    16 16 2 16 8 3 0
n050    16 18 2 16 4 4 0
n051    18 12 2 18 6 4 -1 0
n052    18 14 2 12 8 4 -1 0
n053    18 18 2 16 8 1 16 16 3 -1 0
n054    18 18 1 18 6 1 -2 0
n055    12 18 1 14 8 1 -2 0
n056    16 12 1 -2 0
n057    20 20 2 20 2 1 0
n058    20 18 2 2 20 1 0
n059    20 6 2 10 20 1 0
n060    16 20 2 14 20 1 20 14 1 0
END.
```

80 nets on 21*21 FPGA (100 UPMs)

```
n001    2 2 2 2 14 1 -1 80
n002    8 6 2 2 18 5 2 6 1 0
n003    2 4 2 2 16 1 4 2 1 0
n004    2 6 2 4 12 1 -1 0
n005    4 2 2 4 14 1 4 6 1 6 4 1 0
n006    4 4 2 4 16 1 6 4 1 -1 0
n007    4 6 2 6 14 1 0
n008    6 4 2 -1 0
n009    4 4 1 4 16 1 6 4 1 -2 0
n010    2 6 1 4 12 1 -2 0
n011    2 2 1 2 14 1 -2 0
n012    2 2 1 2 14 1 -2 0
n013    2 2 1 2 14 1 -2 0
n014    2 6 1 4 12 1 -2 0
n015    2 8 2 8 12 1 8 8 1 8 6 1 0
n016    4 8 2 4 12 1 4 4 1 4 6 1 6 4 1 6 6 1 8 6 1 8 8 1 0
n017    6 2 2 -1 0
n018    6 2 1 8 14 1 -2 0
n019    6 6 2 6 18 3 0
n020    6 8 2 6 14 4 0
n021    8 2 2 8 16 4 -1 0
n022    8 4 2 2 18 4 -1 0
n023    8 8 2 6 18 1 6 6 3 -1 0
n024    8 8 1 8 16 1 -2 0
n025    2 8 1 4 18 1 -2 0
n026    6 2 1 -2 0
n027    10 10 2 10 2 1 10 4 1 0
n028    10 8 2 10 2 1 10 6 1 0
n029    10 6 2 10 2 1 10 10 1 0
n030    10 4 2 10 2 1 10 20 1 0
n031    12 12 2 12 4 1 -1 81
n032    18 16 2 12 8 5 12 16 1 0
n033    12 14 2 12 6 1 14 12 1 0
n034    12 16 2 14 2 1 -1 0
n035    14 12 2 14 4 1 14 16 1 16 14 1 0
n036    14 14 2 14 6 1 16 14 1 -1 0
n037    14 16 2 16 4 1 0
n038    16 14 2 -1 0
n039    14 14 1 14 6 1 16 14 1 -2 0
n040    12 16 1 14 2 1 -2 0
n041    12 12 1 12 4 1 -2 0
n042    12 12 1 12 4 1 -2 0
n043    12 12 1 12 4 1 -2 0
n044    12 16 1 14 2 1 -2 0
```

```
n045    12 18 2 18 2 1 18 18 1 18 16 1 0
n046    14 18 2 14 2 1 14 14 1 14 16 1 16 14 1 16 16 1 18
        16 1 18 18 1 0
n047    16 12 2 -1 0
n048    16 12 1 18 4 1 -2 0
n049    16 16 2 16 8 3 0
n050    16 18 2 16 4 4 0
n051    18 12 2 18 6 4 -1 0
n052    18 14 2 12 8 4 -1 0
n053    18 18 2 16 8 1 16 16 3 -1 0
n054    18 18 1 18 6 1 -2 0
n055    12 18 1 14 8 1 -2 0
n056    16 12 1 -2 0
n057    20 20 2 20 2 1 0
n058    20 18 2 2 20 1 0
n059    20 6 2 10 20 1 0
n060    16 20 2 14 20 1 20 14 1 0
n061    12 2 2 12 14 1 -1 80
n062    18 6 2 12 18 5 20 6 1 0
n063    12 4 2 12 16 1 4 2 1 0
n064    12 6 2 14 12 1 -1 0
n065    14 2 2 14 14 1 4 16 1 16 4 1 0
n066    14 4 2 14 16 1 6 14 1 -1 0
n067    14 6 2 16 14 1 0
n068    16 4 2 -1 0
n069    14 4 1 14 16 1 6 4 1 -2 0
n070    12 6 1 14 12 1 -2 0
n071    12 2 1 12 14 1 -2 0
n072    12 2 1 12 14 1 -2 0
n073    12 2 1 12 14 1 -2 0
n074    12 6 1 14 14 1 -2 0
n075    12 8 2 18 12 1 18 18 1 18 16 1 0
n076    16 2 2 -1 0
n077    16 2 1 -2 0
n078    20 8 2 20 2 1 10 6 1 0
n079    2 16 2 4 12 1 -1 0
n080    6 14 2 -1 0
END.
```

REFERENCES

[1] Aladdin A. M. Amin and Habib Youssef. The design and development of a new family of nonvolatile SRAM-based field programmable gate arrays (FPGAs). Project Description, KACST project number AR-14-67, 1995.

[2] B. T. Preas and M. J. Lorenzetti. *Physical Design Automation of VLSI Systems.* The Benjamin Cummings Publisher, CA, 1988.

[3] B. Tseng, J. Rose, and S. Brown. "Improving FPGA Routing Architecture using Architecture and CAD Interactions", *in Proc ICCD-92* pp. 619-697.

[4] Eric Q. Kang, Rung-Bin Lin, and Eugene Shragowitz. "Fuzzy Logic Approach to VLSI Placement", *IEEE Transaction on Very Large Scale Integration (VLSI) Systems*, Vol. 2, No. 4 December 1994, pp. 489-501

[5] Eugene Shragowitz, Habib Youssef, Sadiq M. Sait, and Hakim Adiche. *Fuzzy Genetic Algorithm for Floorplan Design* Proceedings of SPIE's International Symposium on Optical Science, Engineering, and Instrumentation, Application of Soft Computing, San Diego, California, 27 July, 1 August 1997.

[6] J. Frankle, "Iterative and Adaptive Slack Allocation for Performance Driven Layout and FPGA Routing" *29th ACM/IEEE Design Automation Conference*, 1992, pp. 536-542.

[7] Jonathan Rose and Stephen Brown. "Flexibility of Interconnection Structures for Field Programmable Gate Array", *IEEE J. Solid State Circuits*, Vol. 26, No. 3 March 1991, pp. 277-281.

[8] K. Roy, "A Bounded Search Algorithm for Segmented Channel Routing," *IEEE Transaction on Computer Aided Design of Integrated Circuit and Systems*, Vol. 12, No.1, January 1993, pp. 79-95.

[9] Kai Zhu and D. F. Wong, "On Channel Segmented Design of Row-Based FPGAs," *1992 International Conference on Computer-Aided Design, ICCAD-92*, pp. 26-29.

[10] Kai Zhu and D. F. Wong, Yao-Wen "Switch Module Design with Application to Two Dimensional Segmentation Design". *International Conference on Computer Aided Design, ICCAD-93*, pp. 480-485.

[11] M. Palczewski. "Plane Parallel A* maze Router and its Application to FPGA." 29th ACM/IEEE Design Automation Conference, 1992, pp 691-697.

[12] Michael J. Alexander and Gabriel Robins, "New Performance-Driven FPGA Routing Algorithms." *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 15, No. 12, December 1996. pp. 1505-1517

[13] Richard Barnett. *Microprocessor System Design Technology*. Sigma Technical Press, 1991.

[14] S. Brown, J. Rose and Z. Vranesic. "A Detailed Router for Field-Programmable Gate Arrays." *IEEE Transaction on Computer-Aided Design*. Vol. 11, No. 5, May 1992. pp. 620-628.

[15] S. Burman, C. Kamalanathan, and N. Sherwani, "New Channel SegmentationModel and associated routing algorithm for high performance FPGAs" *1992 International Conference on Computer-Aided Design, ICCAD-92*, pp 22-25.

[16] S. D. Brown, R. J. Francis, J. Rose, and Z. G. Vranesie. *Field Programmable Gate Array*. Kuluwer Academic Publisher 1992

[17] S. Thakur, Y. Chang D.F. Wong and S. Muthukrishnan. "Algorithms for an FPGA Switch Module Routing Problem with Application to Global Routing." *IEEE Transaction on Computer-Aided Design of Integrated Circuits and Systems* Vol 16, No. 1, January 1997. pp 32-46.

[18] Sadiq M. Sait and Habib Youssef. *VLSI Physical Design Automation: Theory and Practice*. McGraw-Hill Book Company, Europe 1995.

[19] V. Roychowdhury, J. Greene, and Abbas El-Gamal, Segmented Channel Routing for FPGAs and Associated Channel Architecture Issues." *IEEE Transaction on Computer Aided Design of Integrated Circuit and Systems*, Vol. 12, No. 12, November 1993, pp. 1697-1705.

[20] W. Swartz and C. Sechrn "A new Generalized Row-Based Global Router." *1993 International Conference on Computer-Aided Design, ICCAD-93*, pp 480-485.

[21] Y. Sun, T. Wang, C. K. Wong and C. L. Liu. "Routing for symmetric FPGA's and FPIC's". *1993 International Conference on Computer-Aided Design ICCAD-93*. pp 486-490

[22] Y. Sun, T. Wang, C. K. Wong and C. L. Liu. "Routing for symmetric FPGA's and FPIC's". *IEEE Transaction on Computer-Aided Design of Integrated Circuits and Systems* Vol 16, No. 1, January 1997. pp 20-31.

[23] Yu-Liang Wu, Shuji Tsukiyama, and Malgorzata Marek-Sadowska. Graph Based Analysis of 2-D FPGA Routing. *IEEE Transaction on Computer-Aided Design of Integrated Circuits and Systems*. 15(1): 33-44, January 1996

VITA

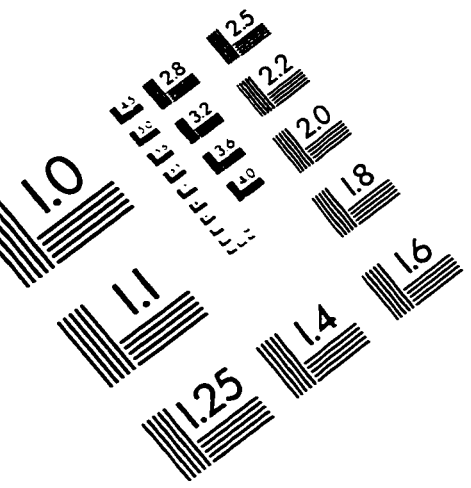
- Talal Mousa Mohammed Al-Kharobi

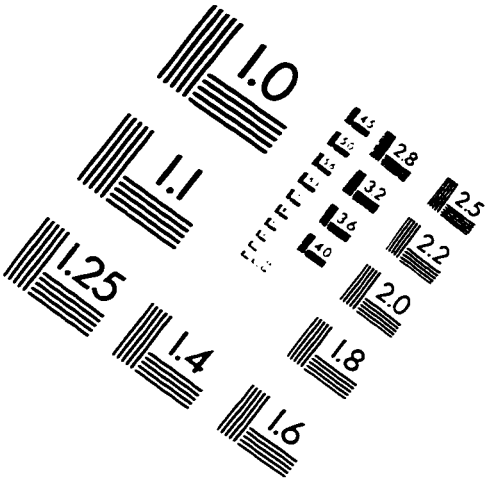
- Received Bachelor's degree in Computer Engineering with high honor from **King Fahd University of Petroleum & Minerals**, Dhahran, Saudi Arabia in July 1993.

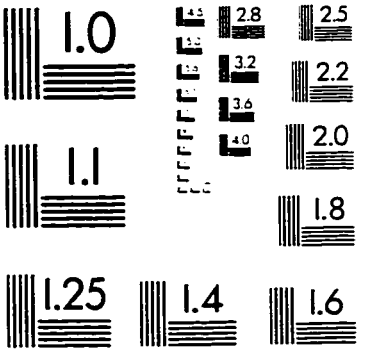
- Working as Graduate Assistant in the Computer Engineering Department at **King Fahd University of Petroleum & Minerals**, Dhahran, Saudi Arabia since September 1993.

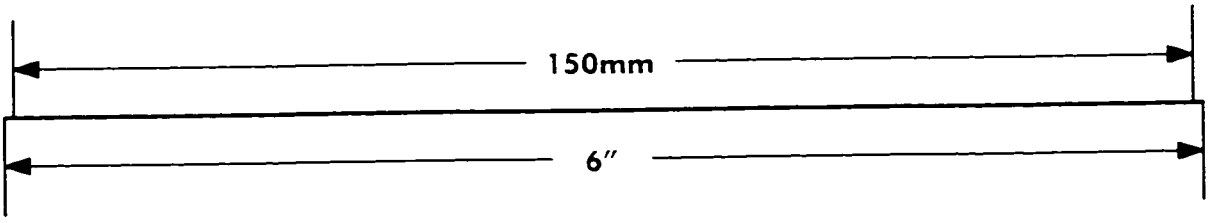
- Completed Master's Degree requirements at **King Fahd University of Petroleum & Minerals**, Dhahran, Saudi Arabia in June 1998.

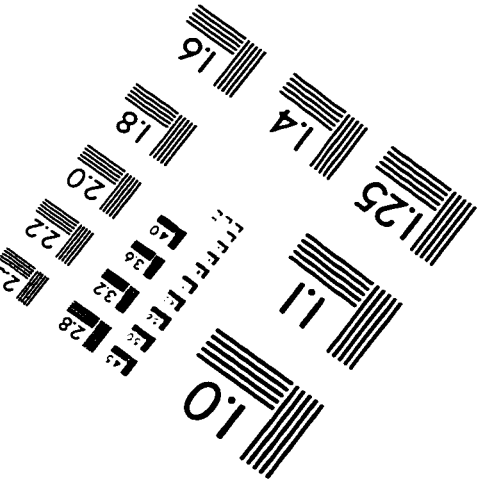
IMAGE EVALUATION
TEST TARGET (QA-3)

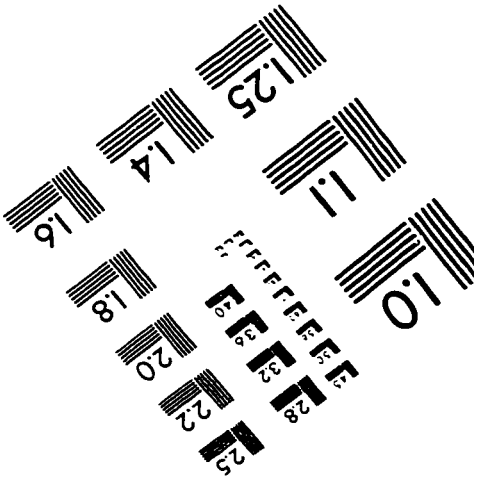












APPLIED IMAGE, Inc
1653 East Main Street
Rochester, NY 14609 USA
Phone: 716/482-0300
Fax: 716/288-5989